



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

- 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.

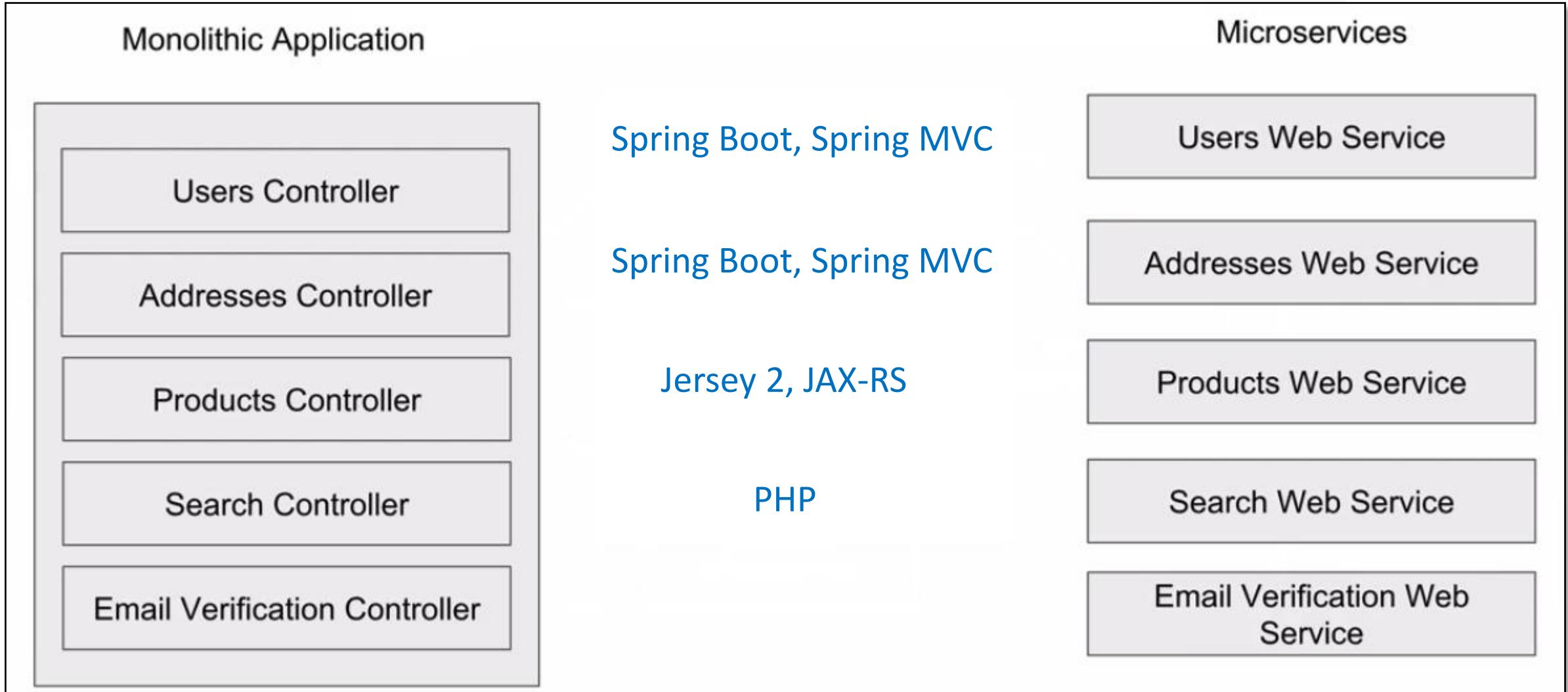


What are Microservice?

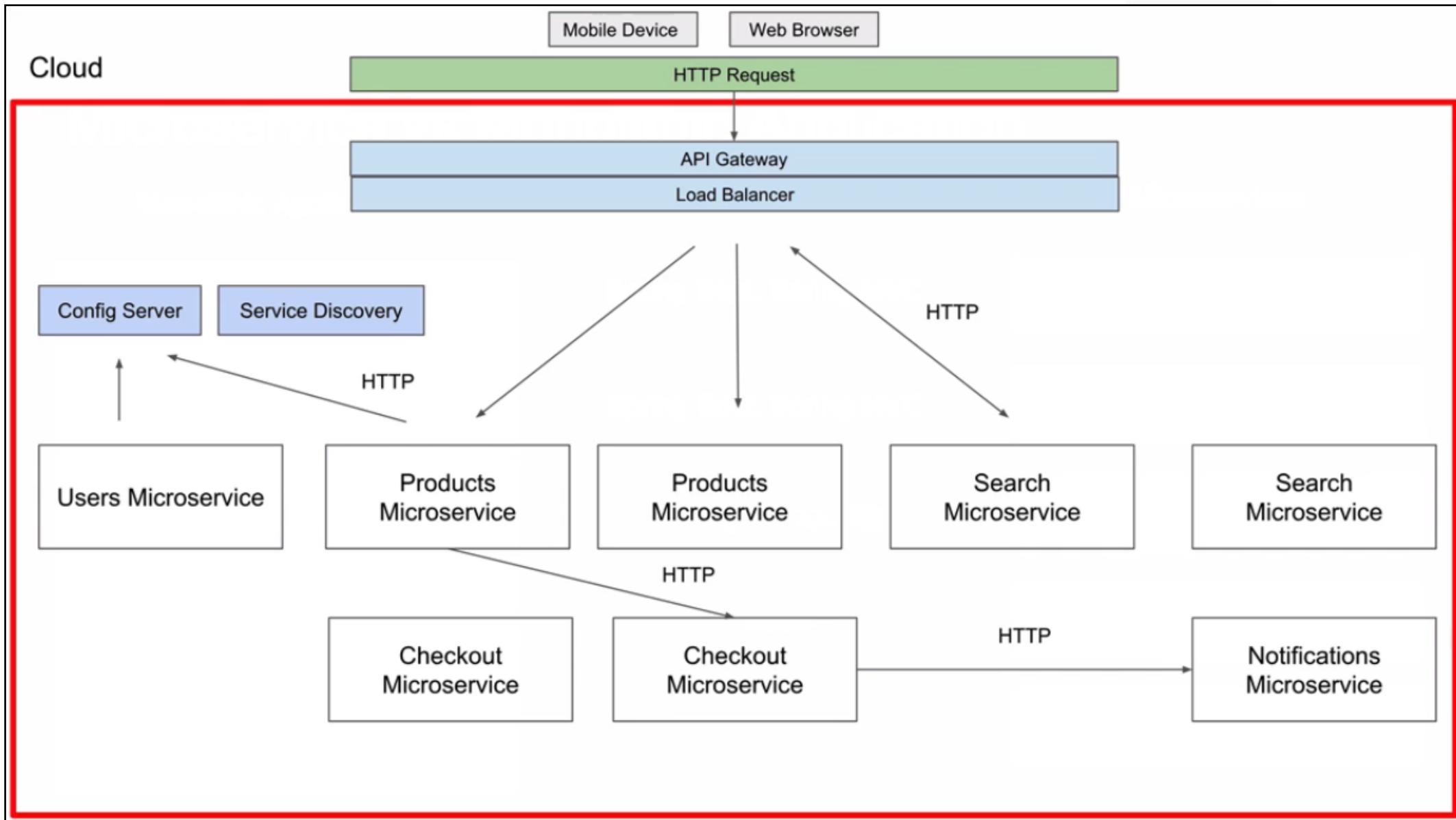
- A Web Service
- Small and Responsible for one thing (Search, Password Reset, Email Verification)
- Configured to work in the cloud and is easily scalable.



Microservice vs Monolithic Application



Application Diagram



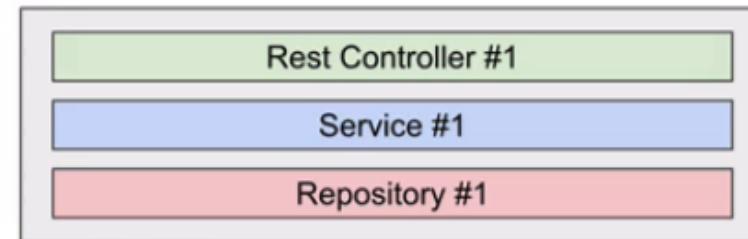


Rest Controller - Services

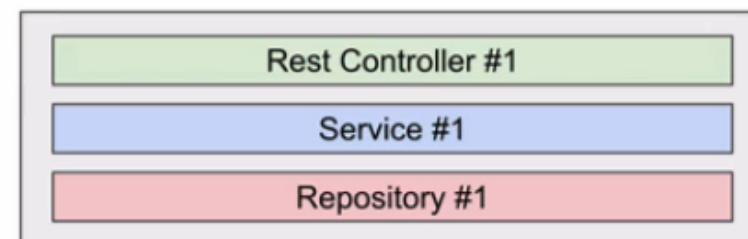
Users



Addresses



Search

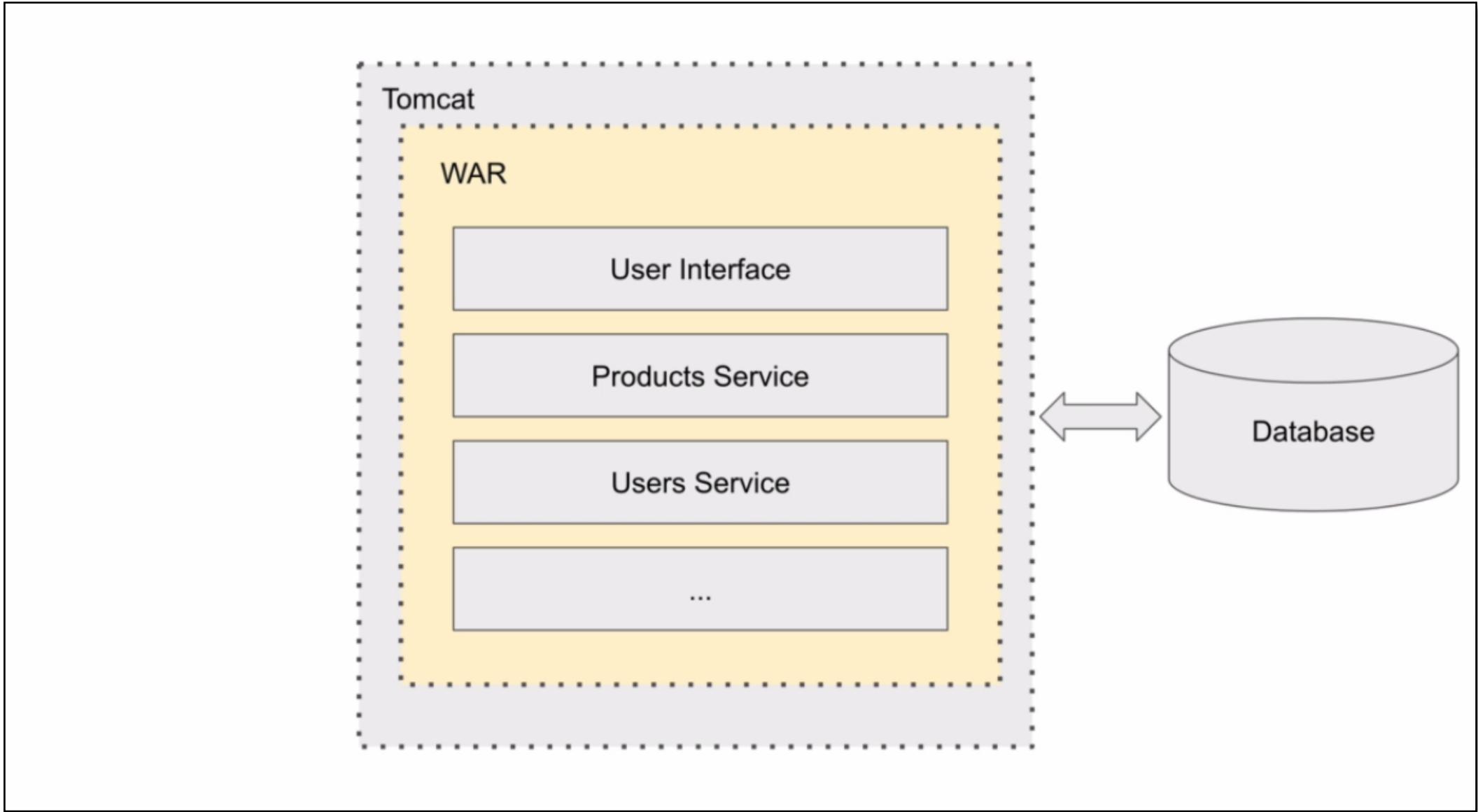




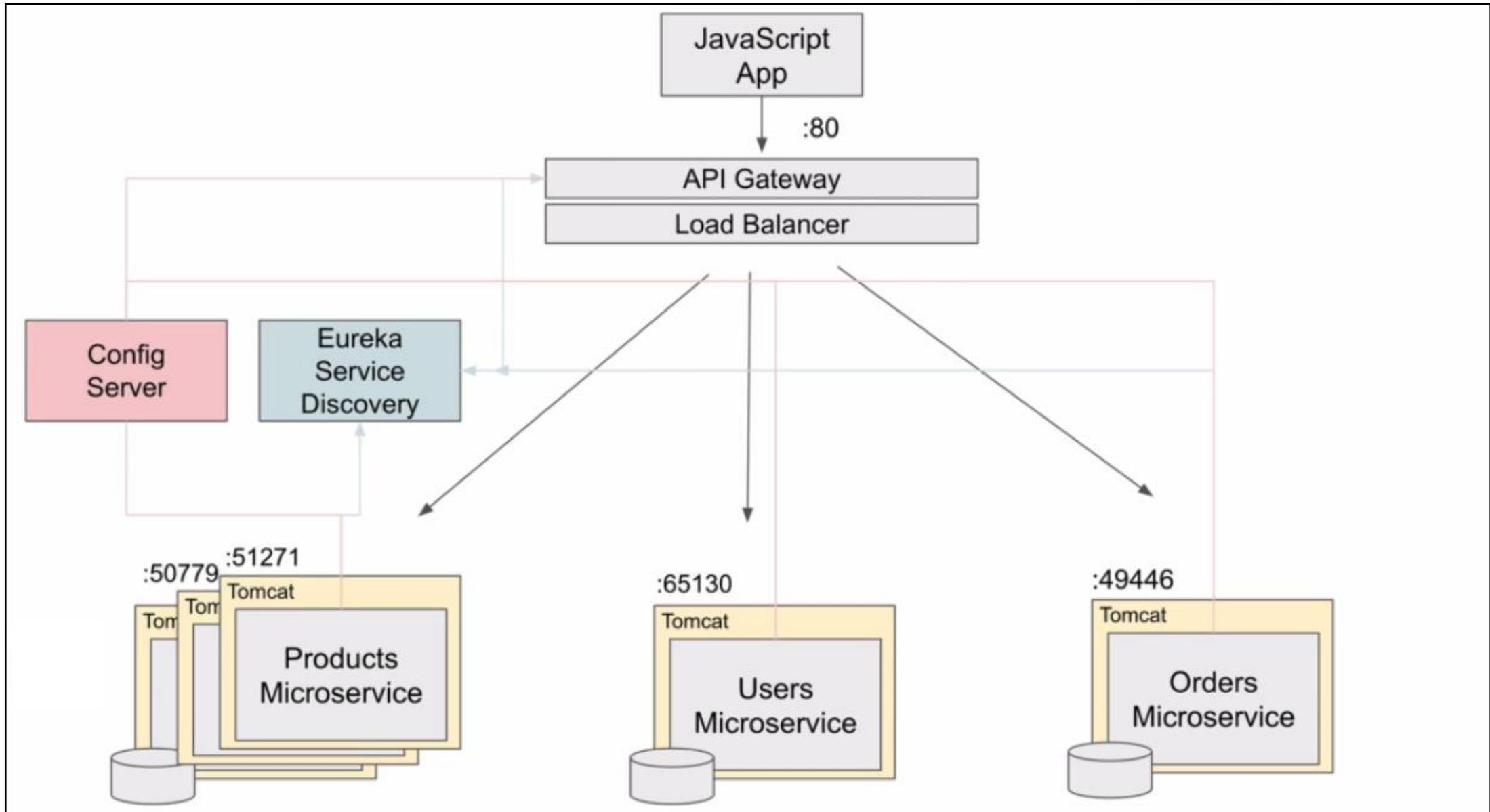
Day - 1

Microservices Architecture Overview

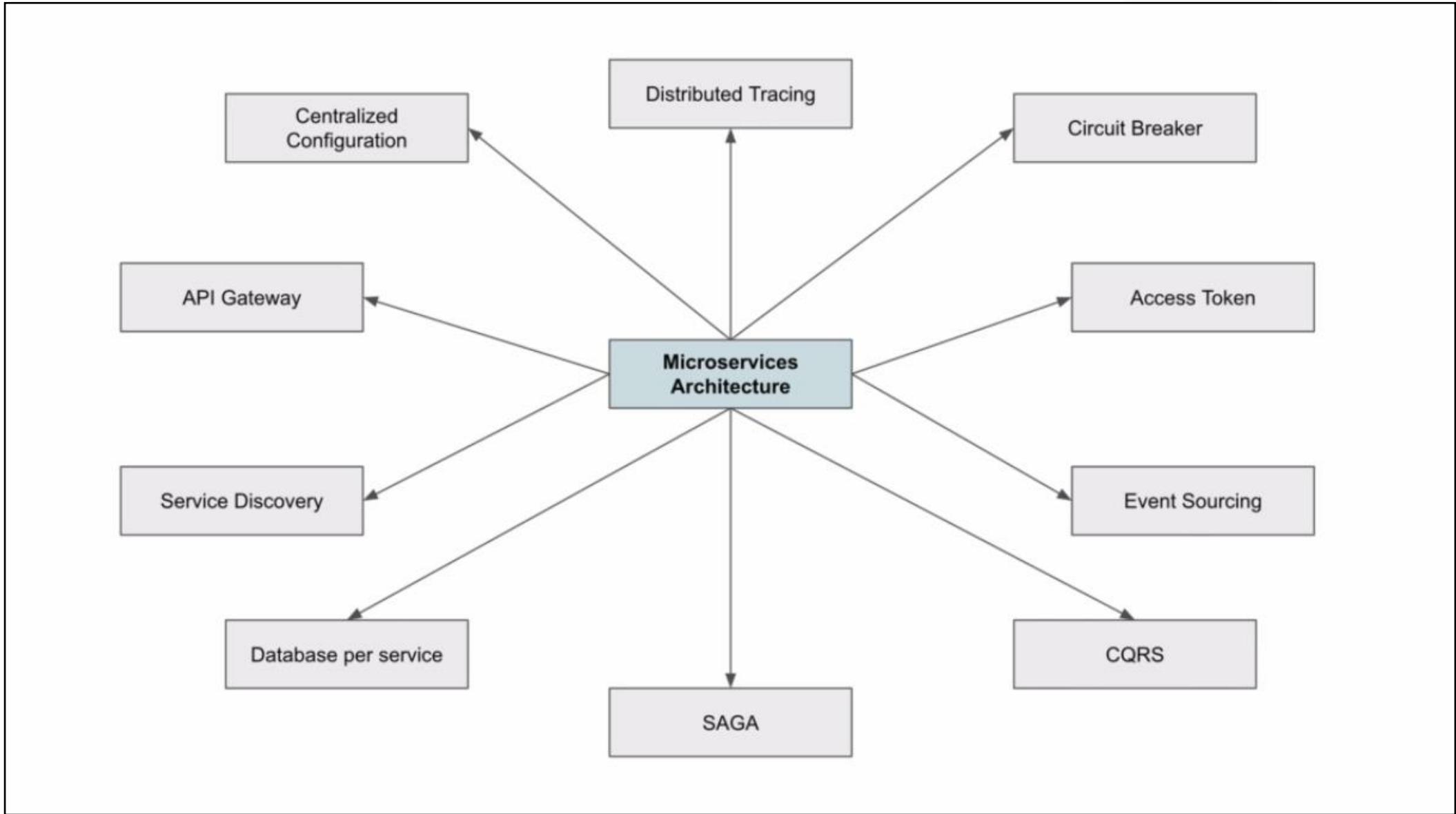
Monolithic Architecture



Microservices Architecture



Pattern Diagram





Day - 1

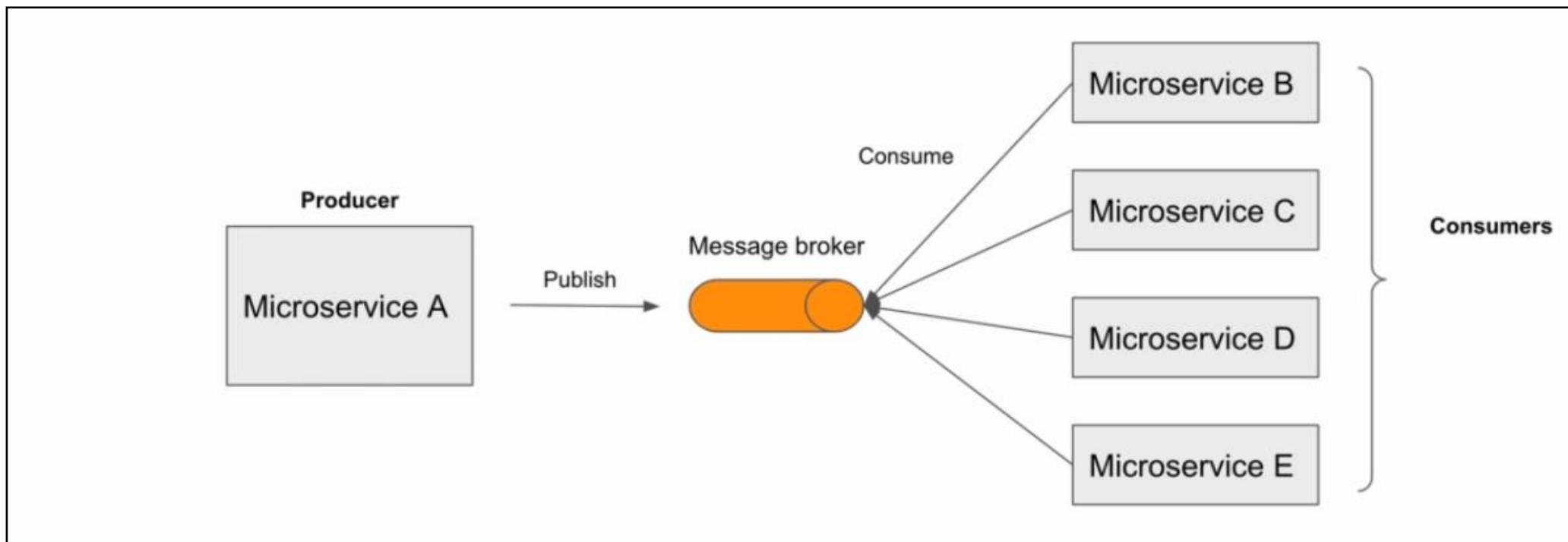
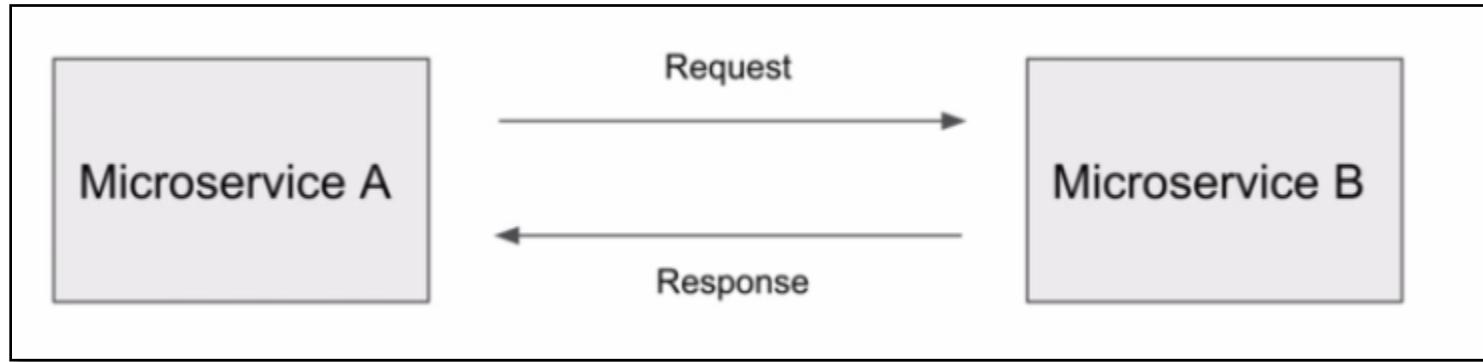
Event-Driven Microservices



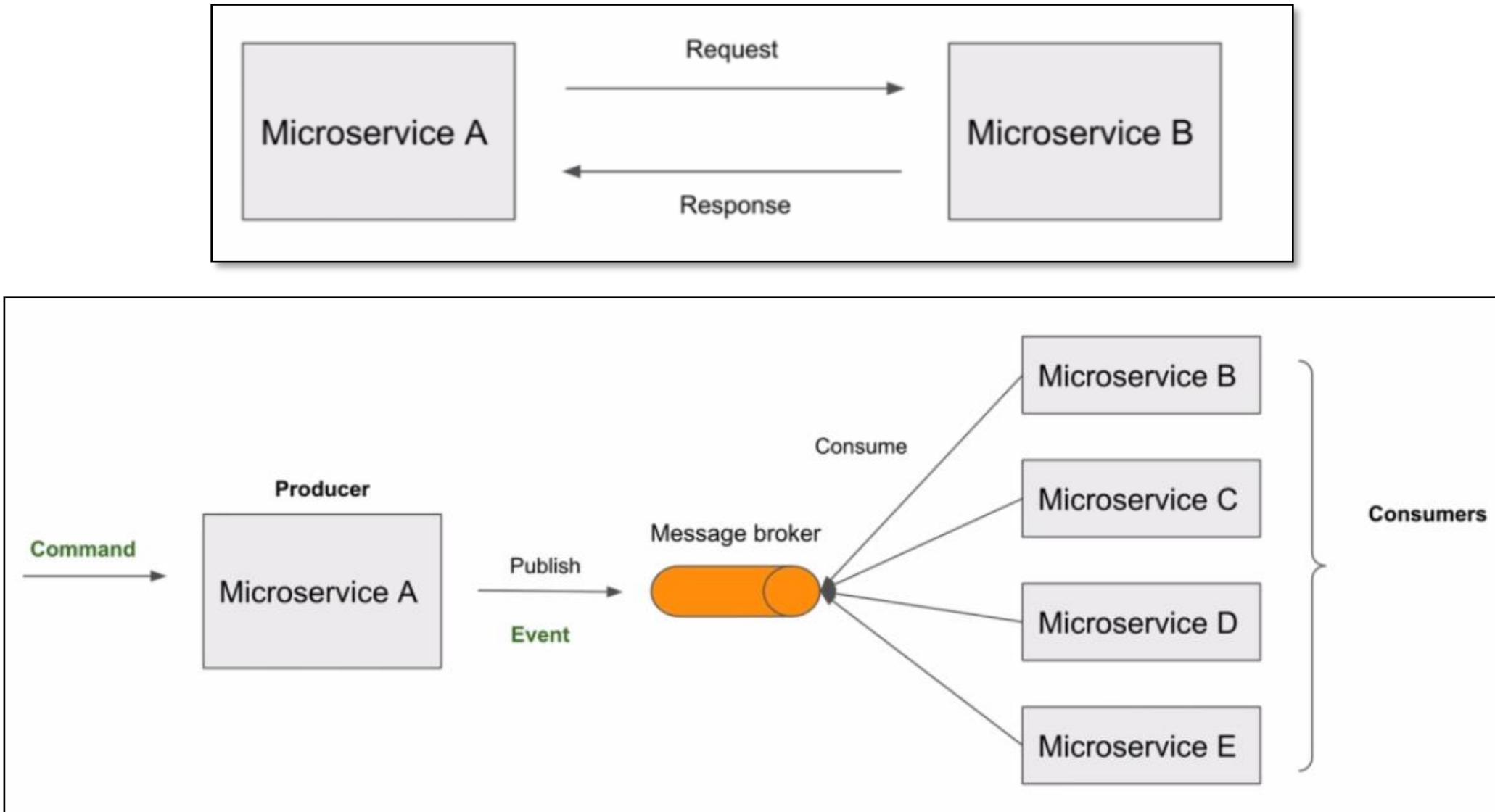
Event-Driven Microservices

- 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.

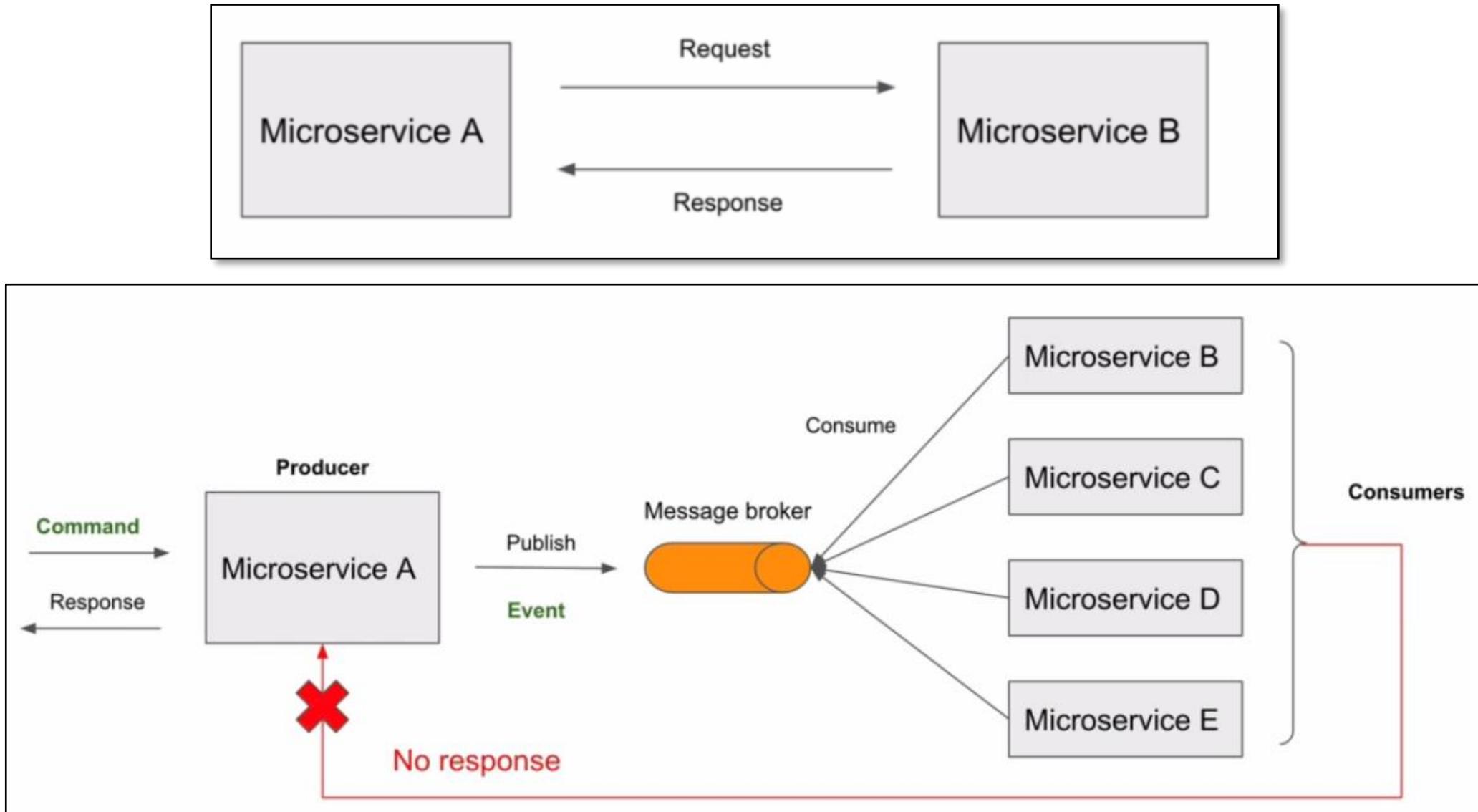
Event-Driven Microservices



Event-Driven Microservices



Event-Driven Microservices





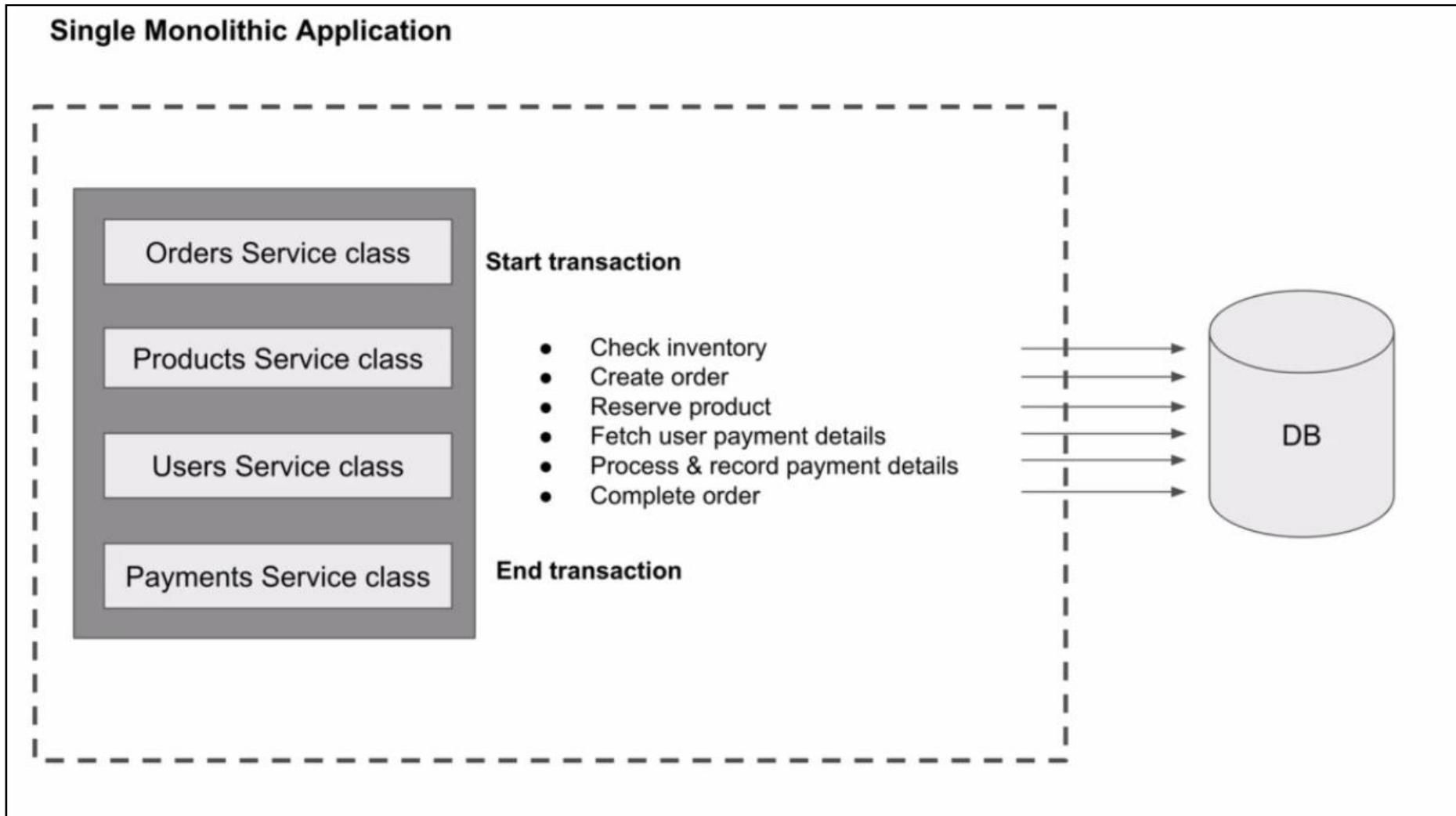
Day - 1

Transactions in Microservices



Transactions in Microservices

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

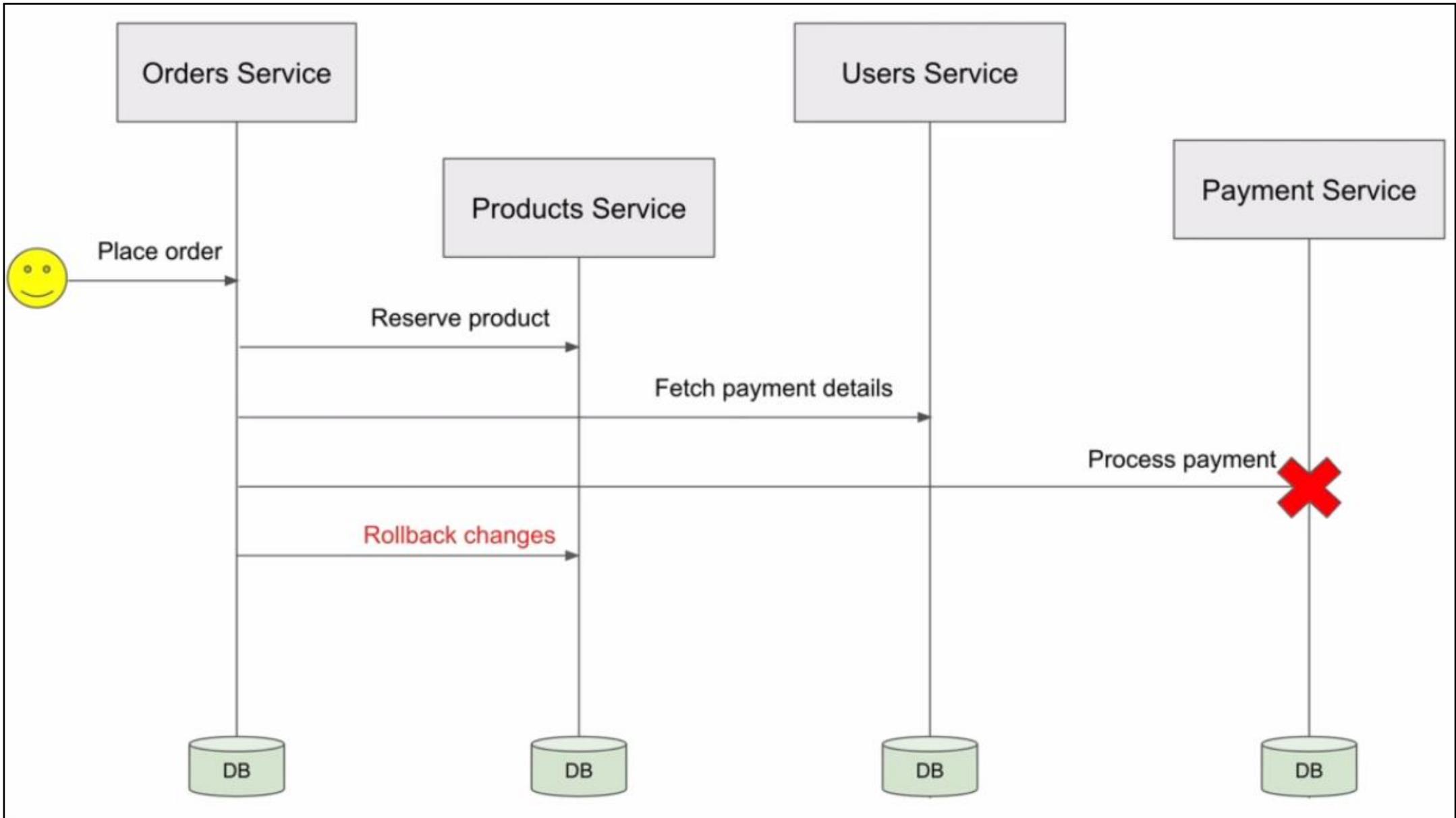




ACID Transactions

- 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

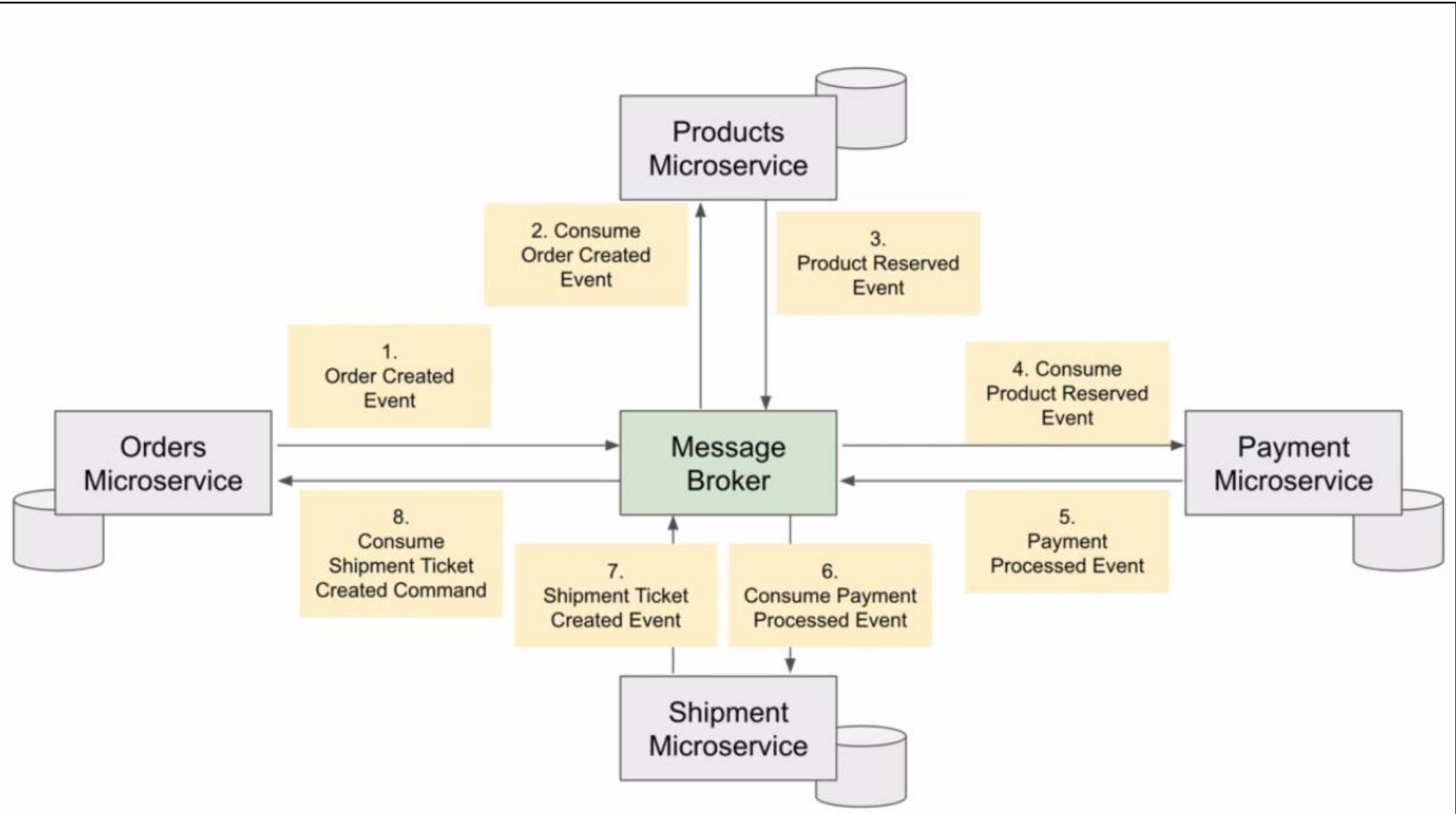
Choreography-Based SAGA



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

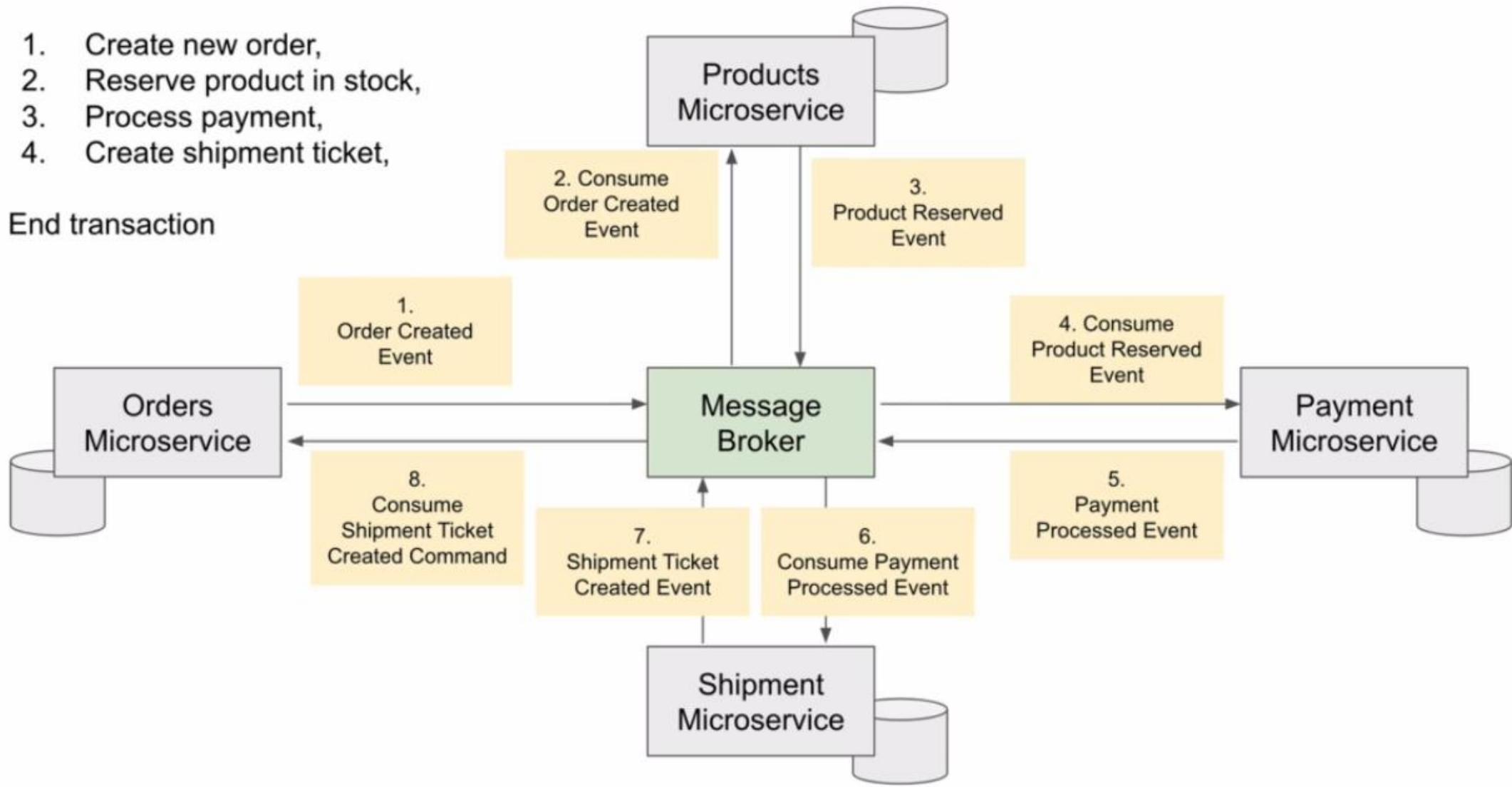


Choreography-Based SAGA

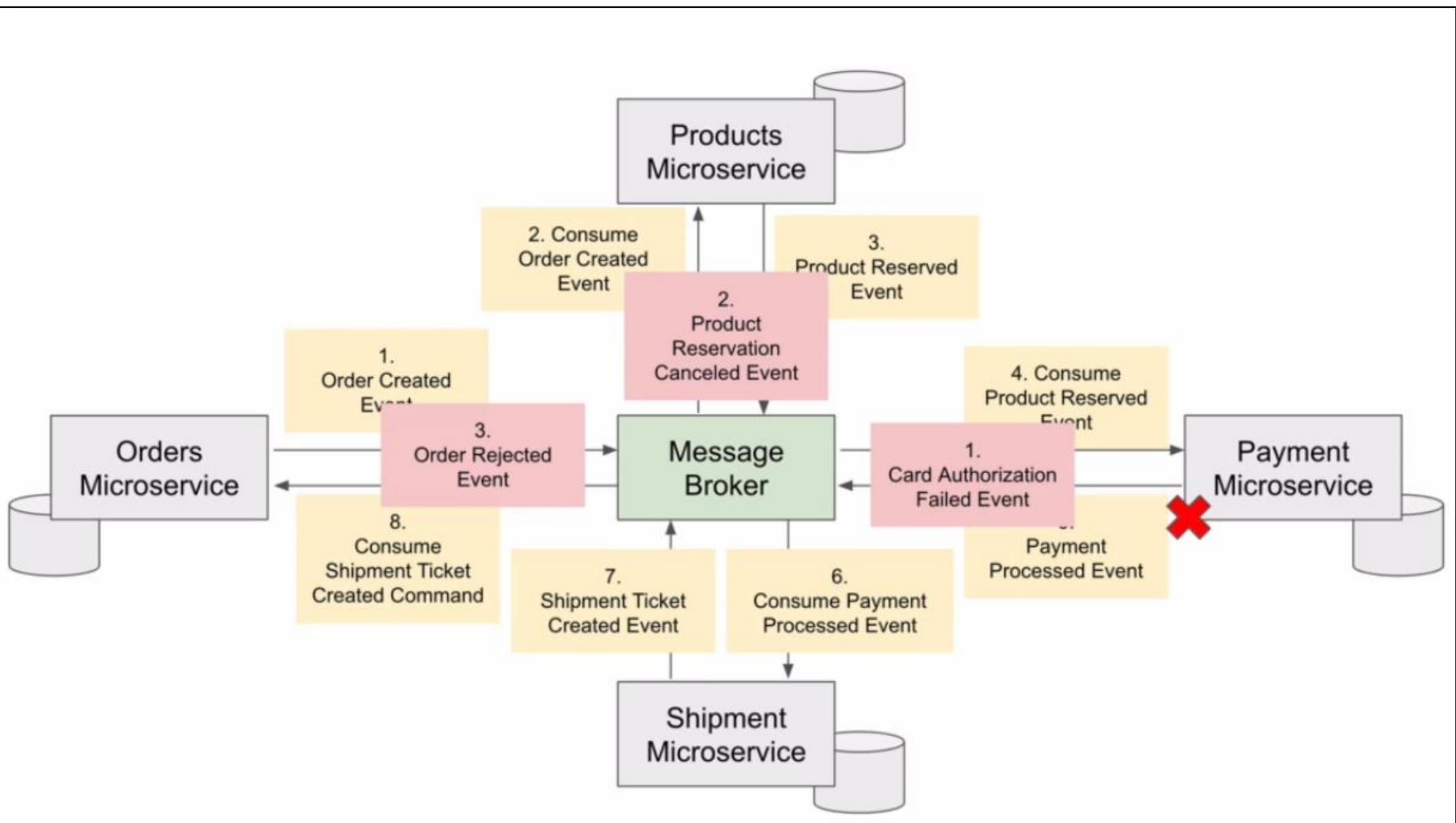
Begin Transaction

1. Create new order,
2. Reserve product in stock,
3. Process payment,
4. Create shipment ticket,

End transaction



Choreography-Based SAGA



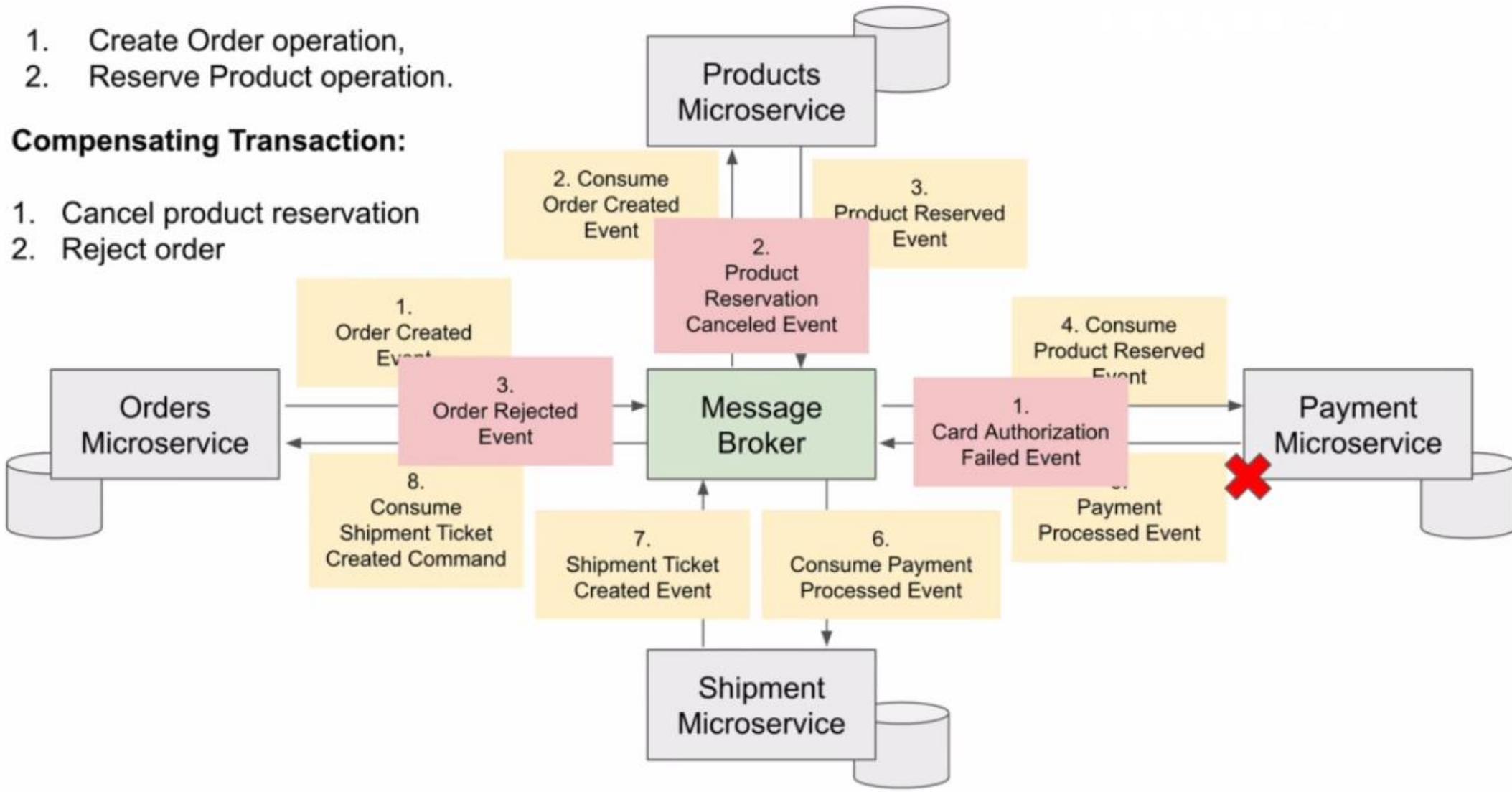
Choreography-Based SAGA

Initial Flow:

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

Compensating Transaction:

1. Cancel product reservation
2. Reject order



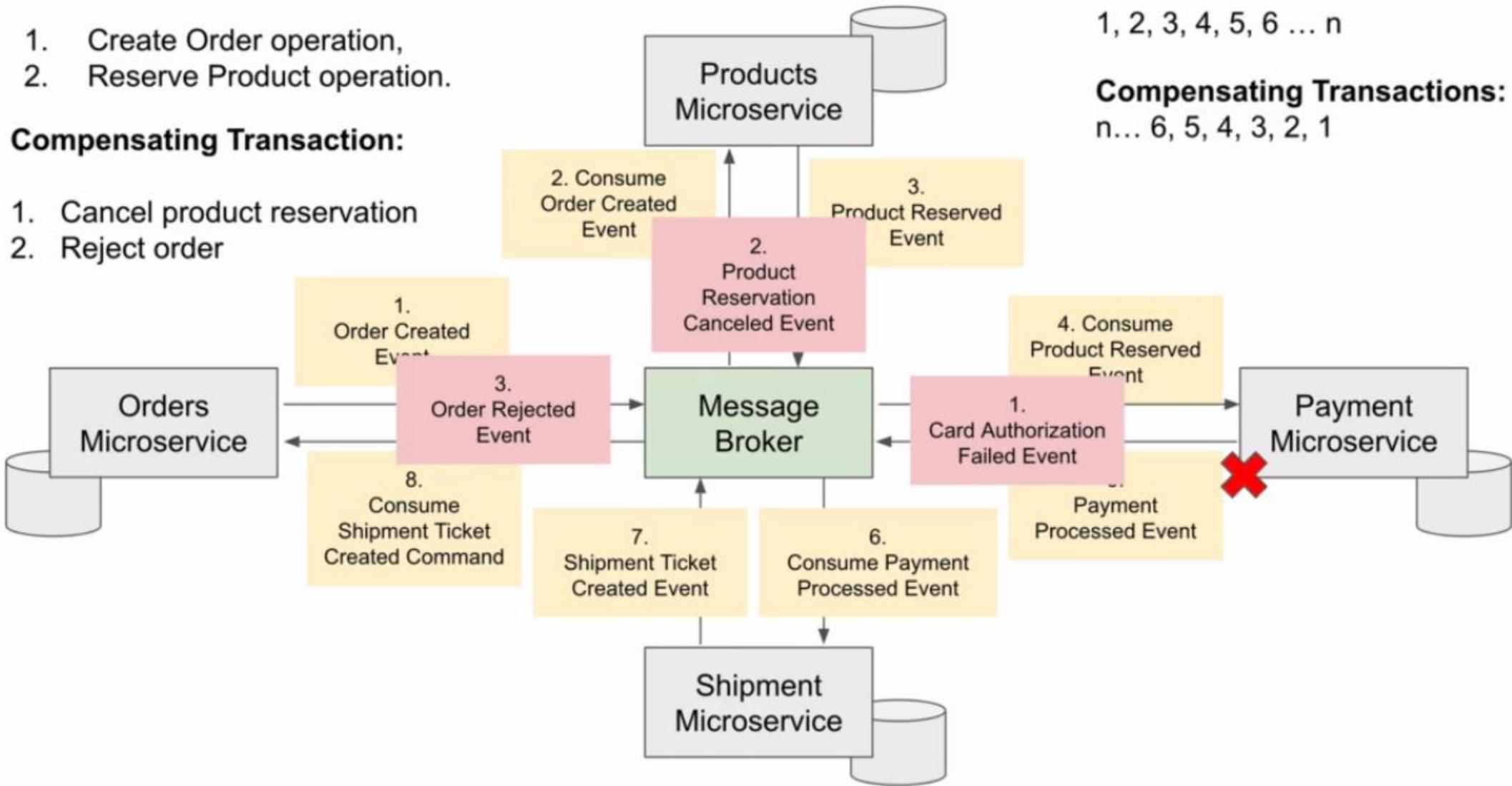
Choreography-Based SAGA

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

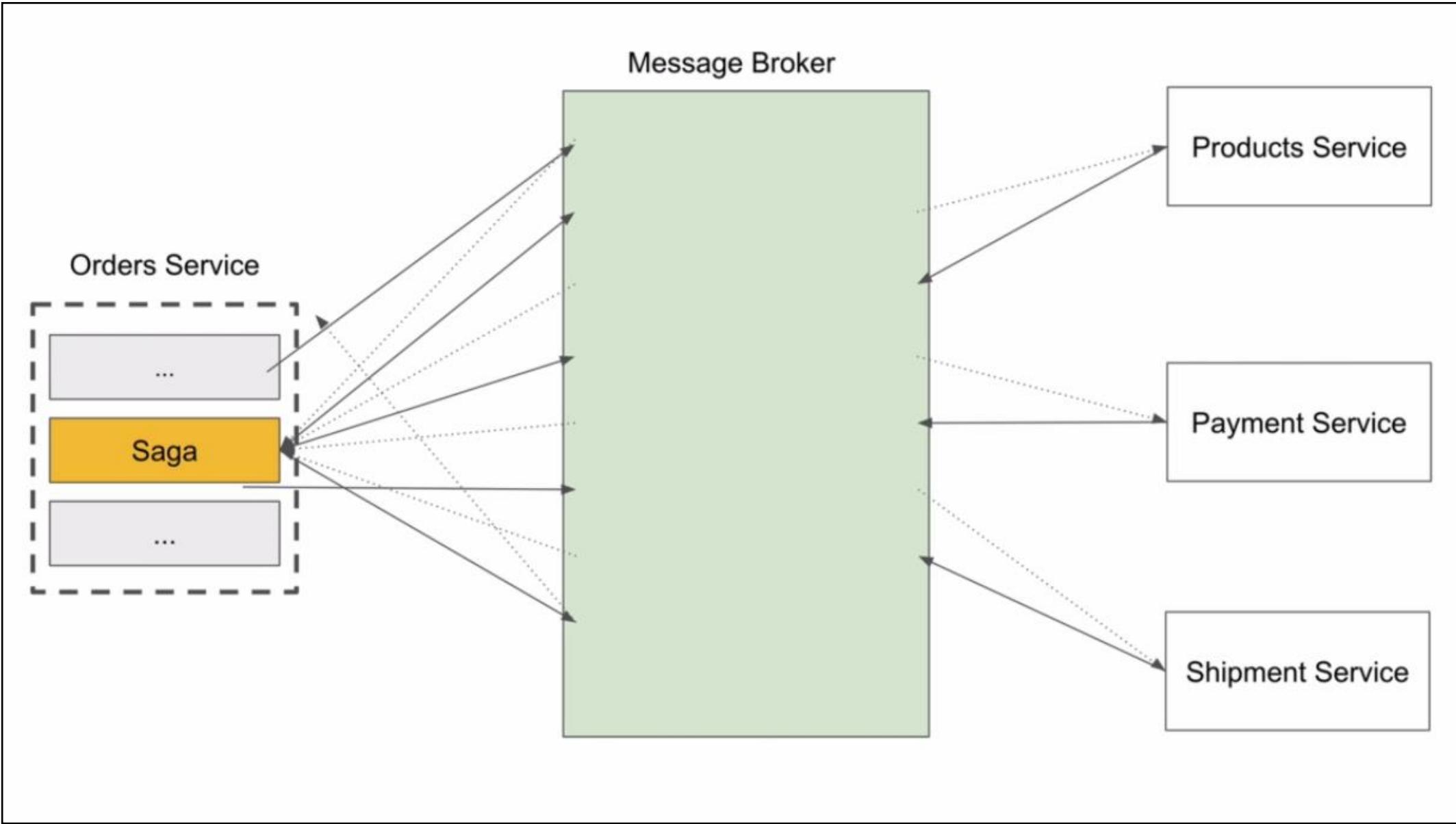
Orchestration-Based SAGA



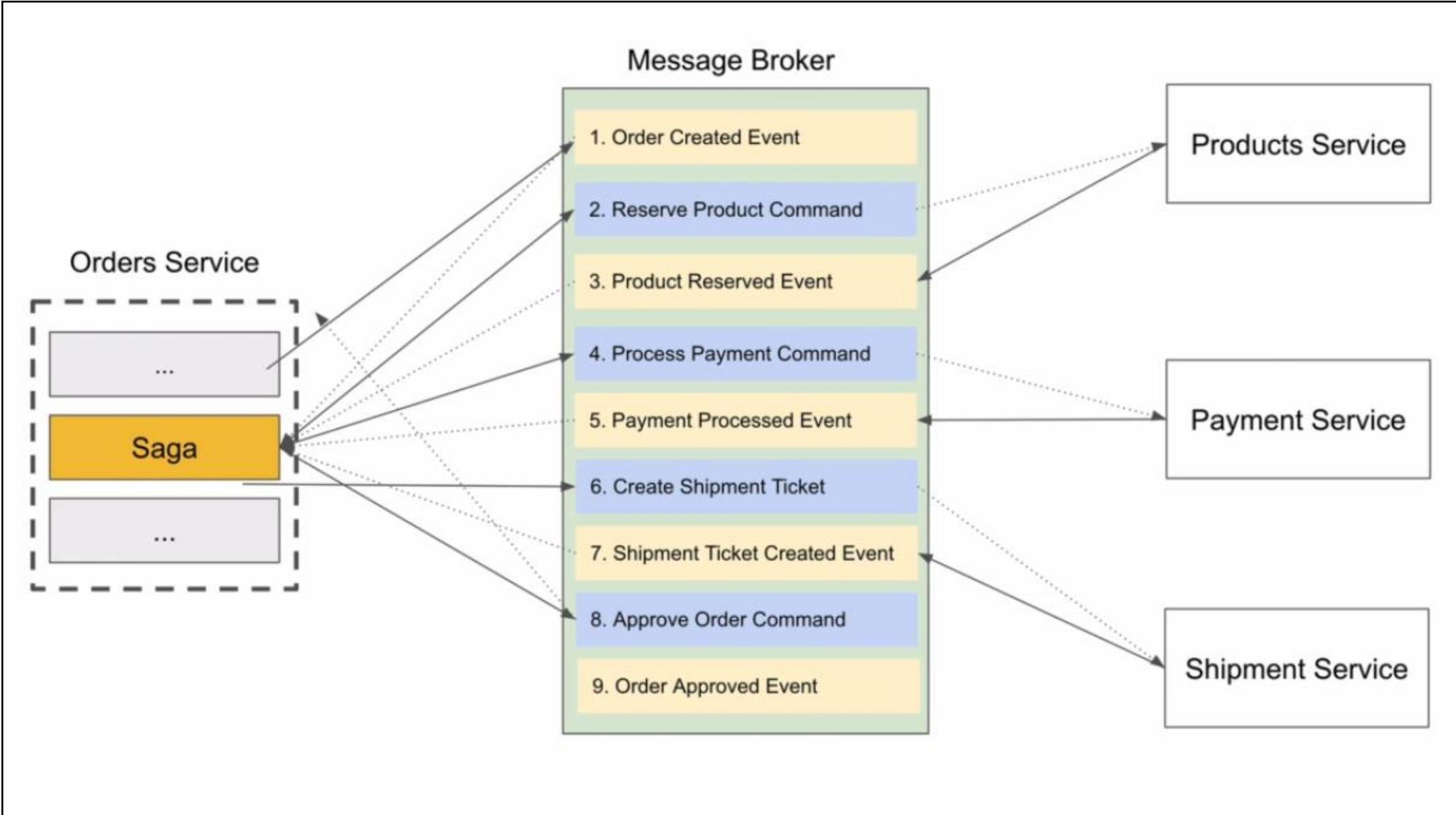
Orchestration-Based SAGA

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

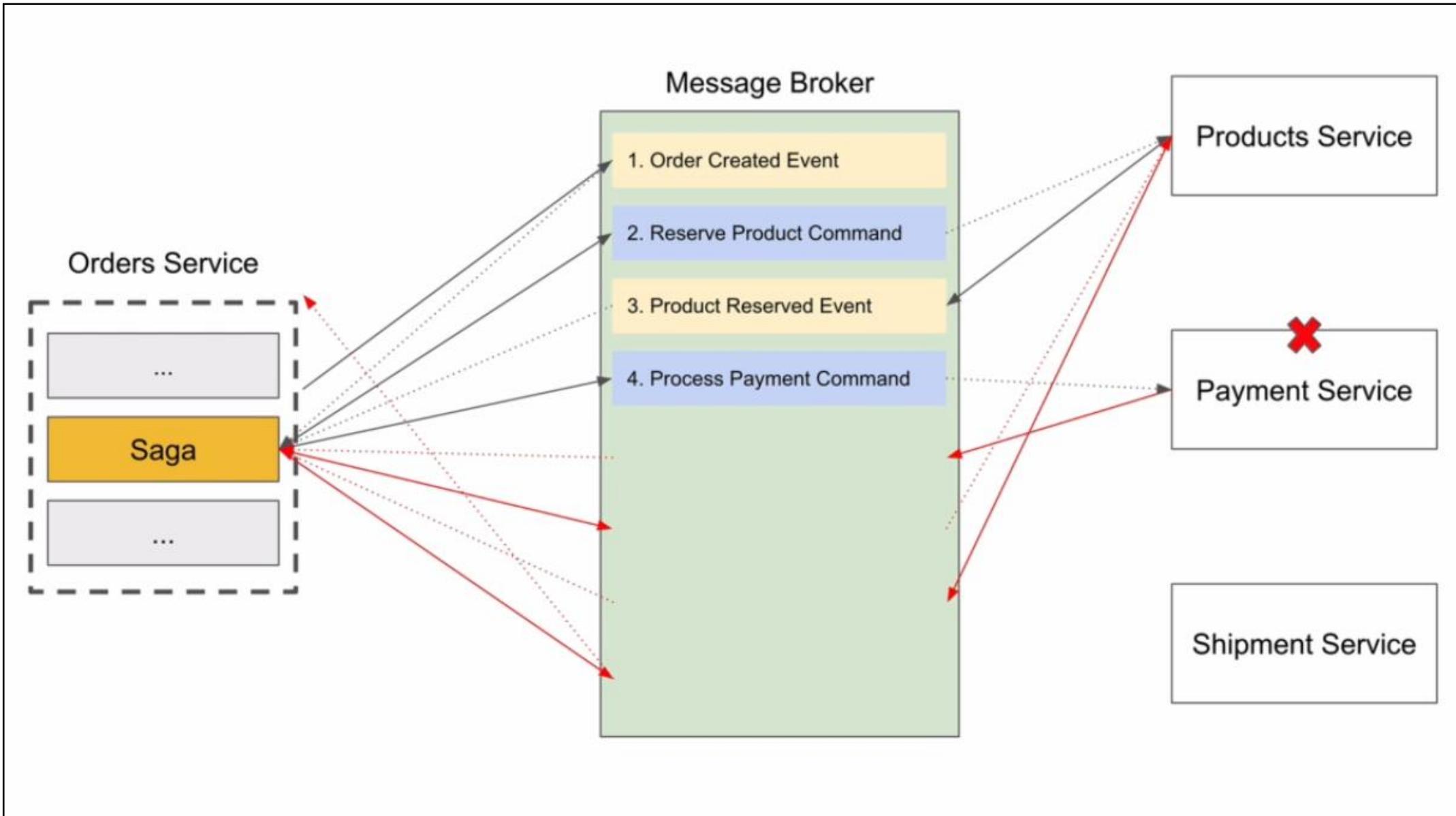
Orchestration-Based SAGA



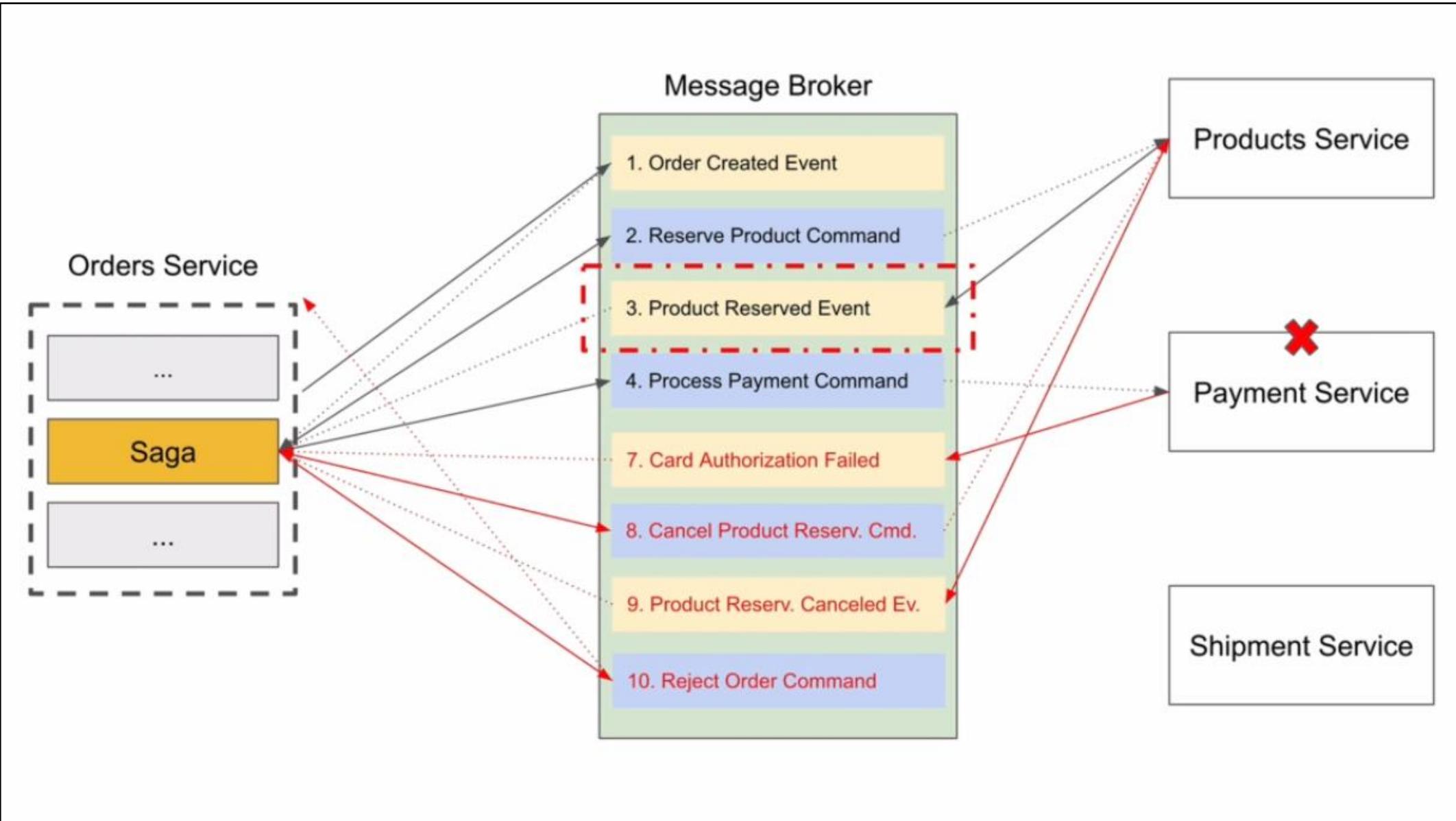
Orchestration-Based SAGA



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.

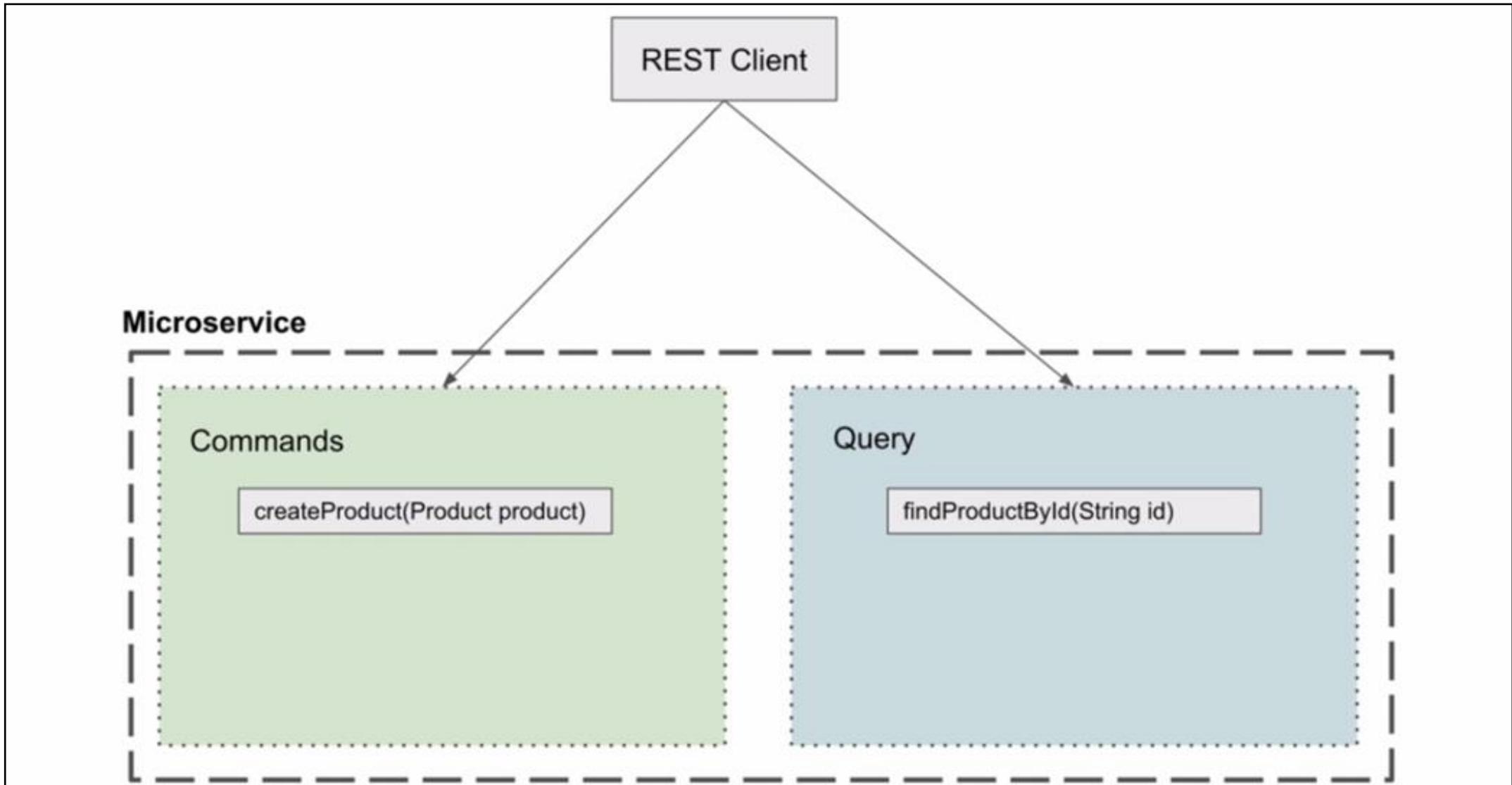


Day - 1

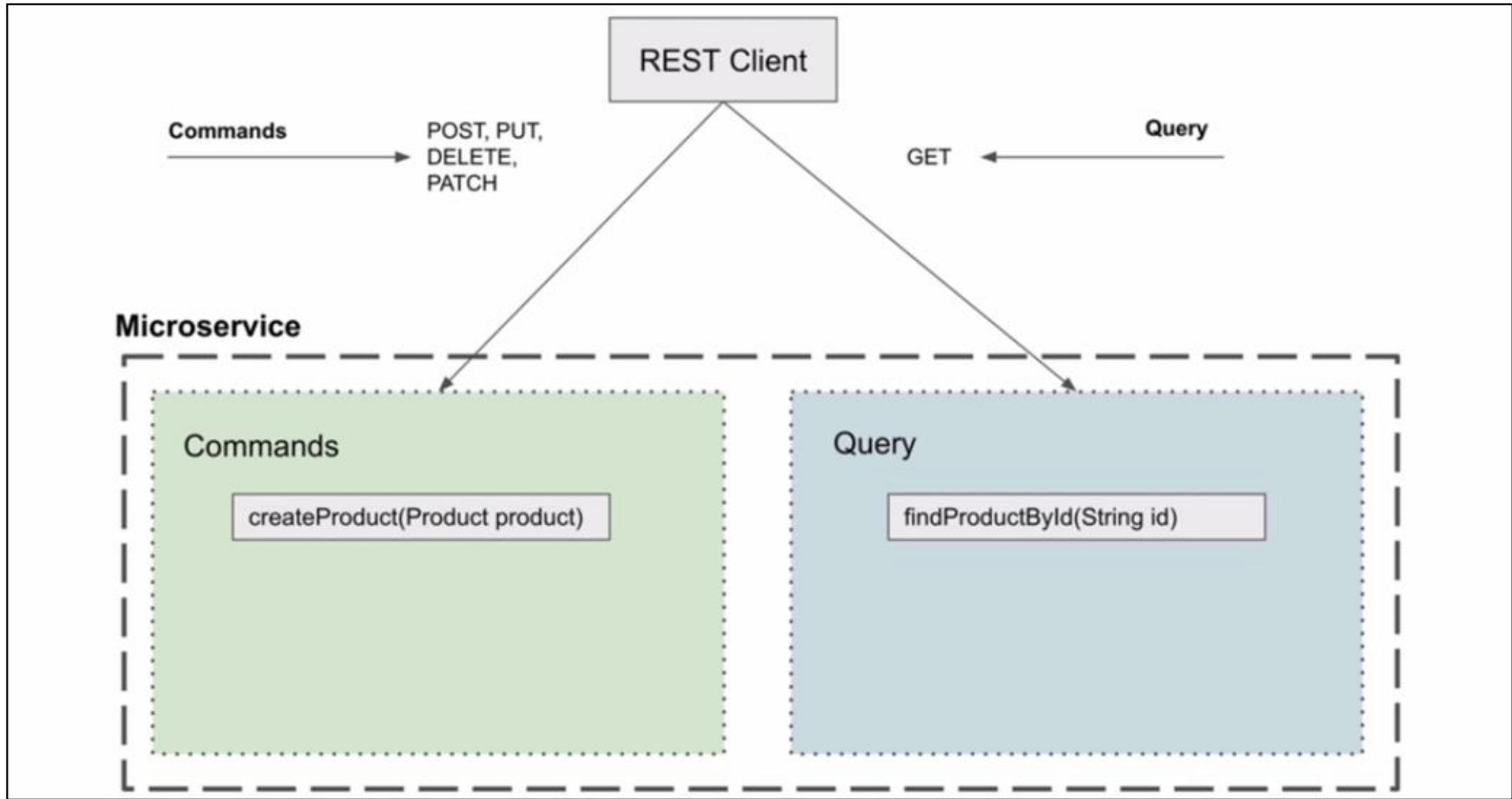
Command Query Responsibility Segregation

- 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.

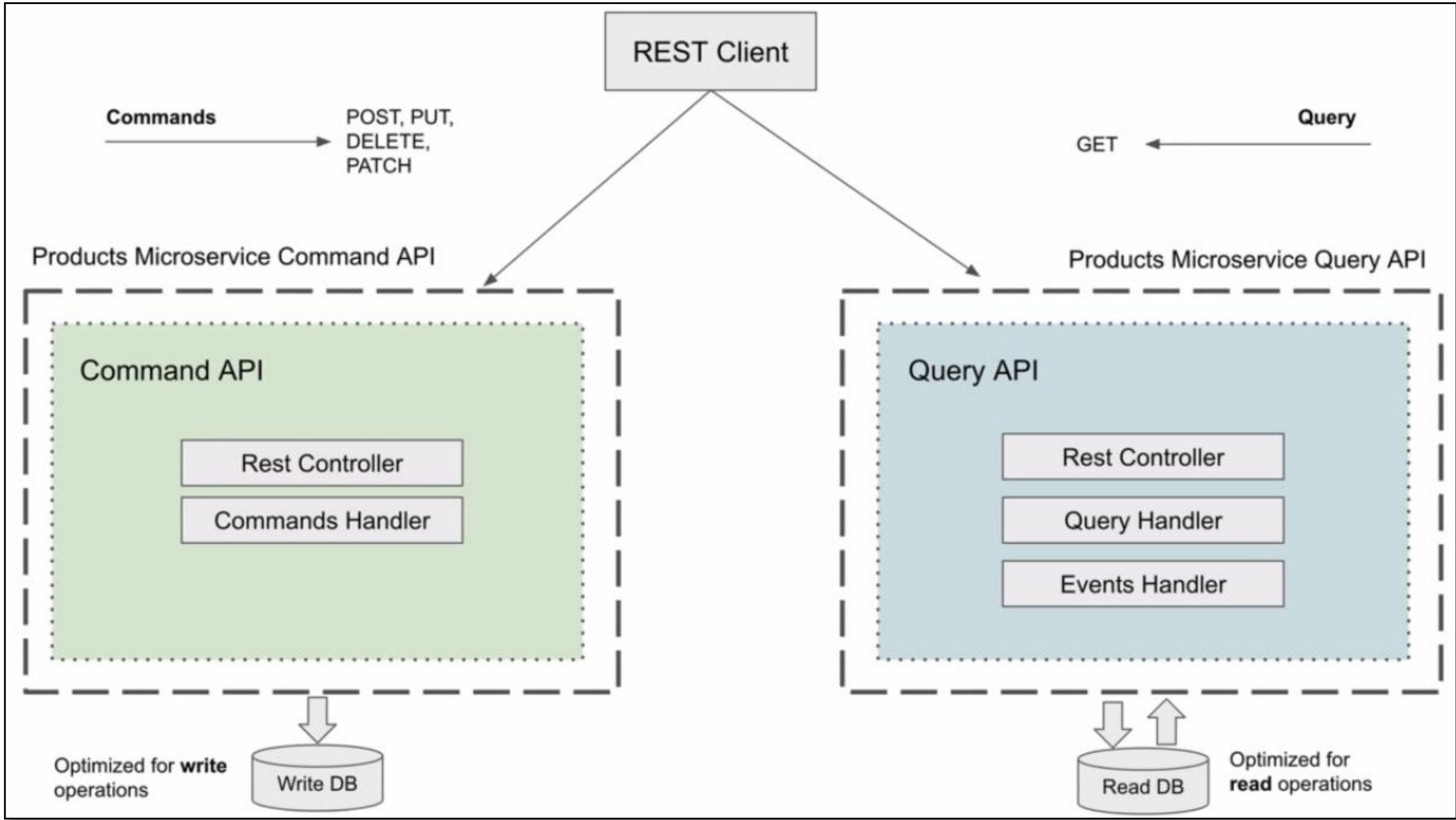
Command Query Responsibility Segregation



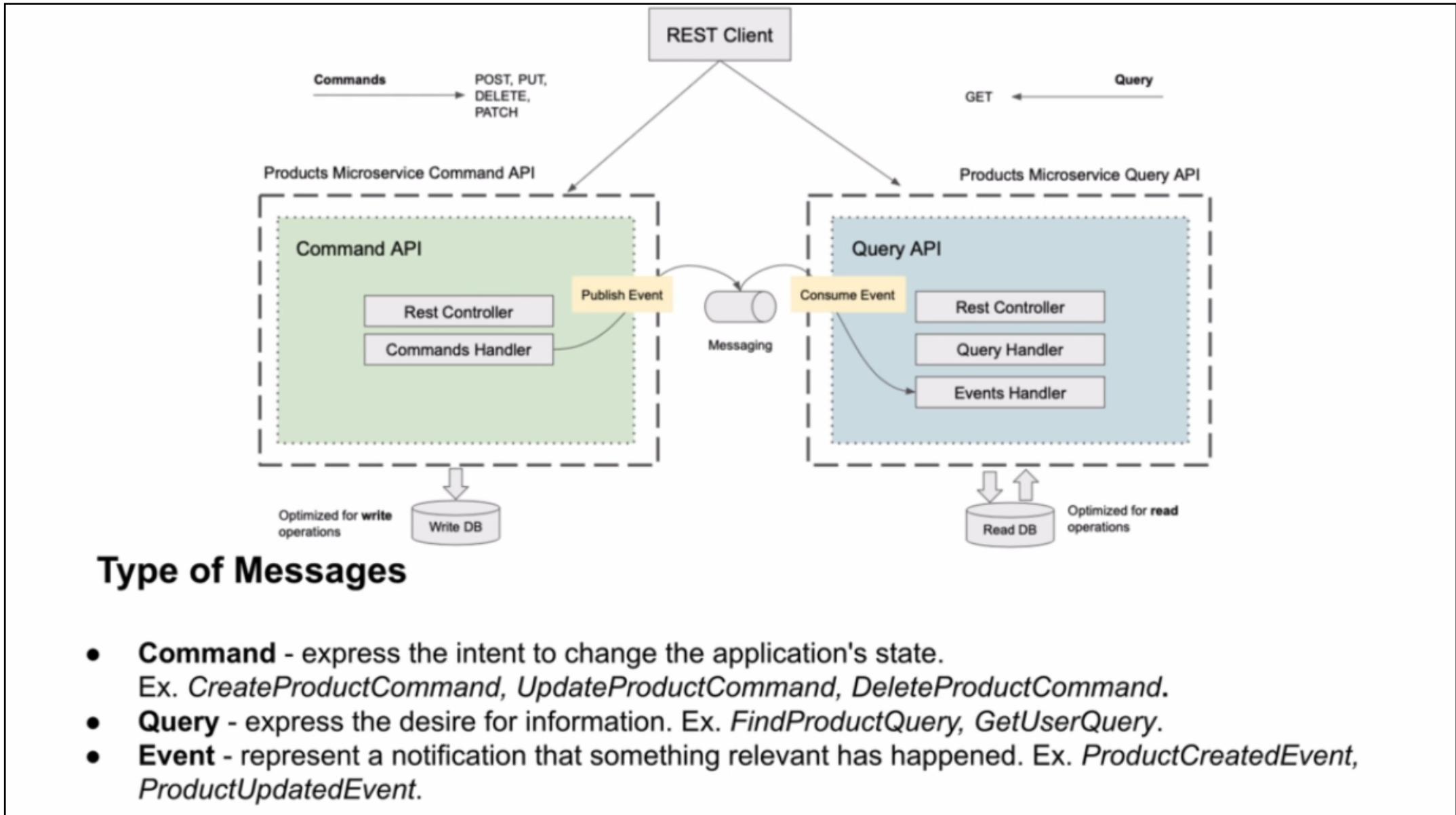
Command vs Query – Controllers



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*.

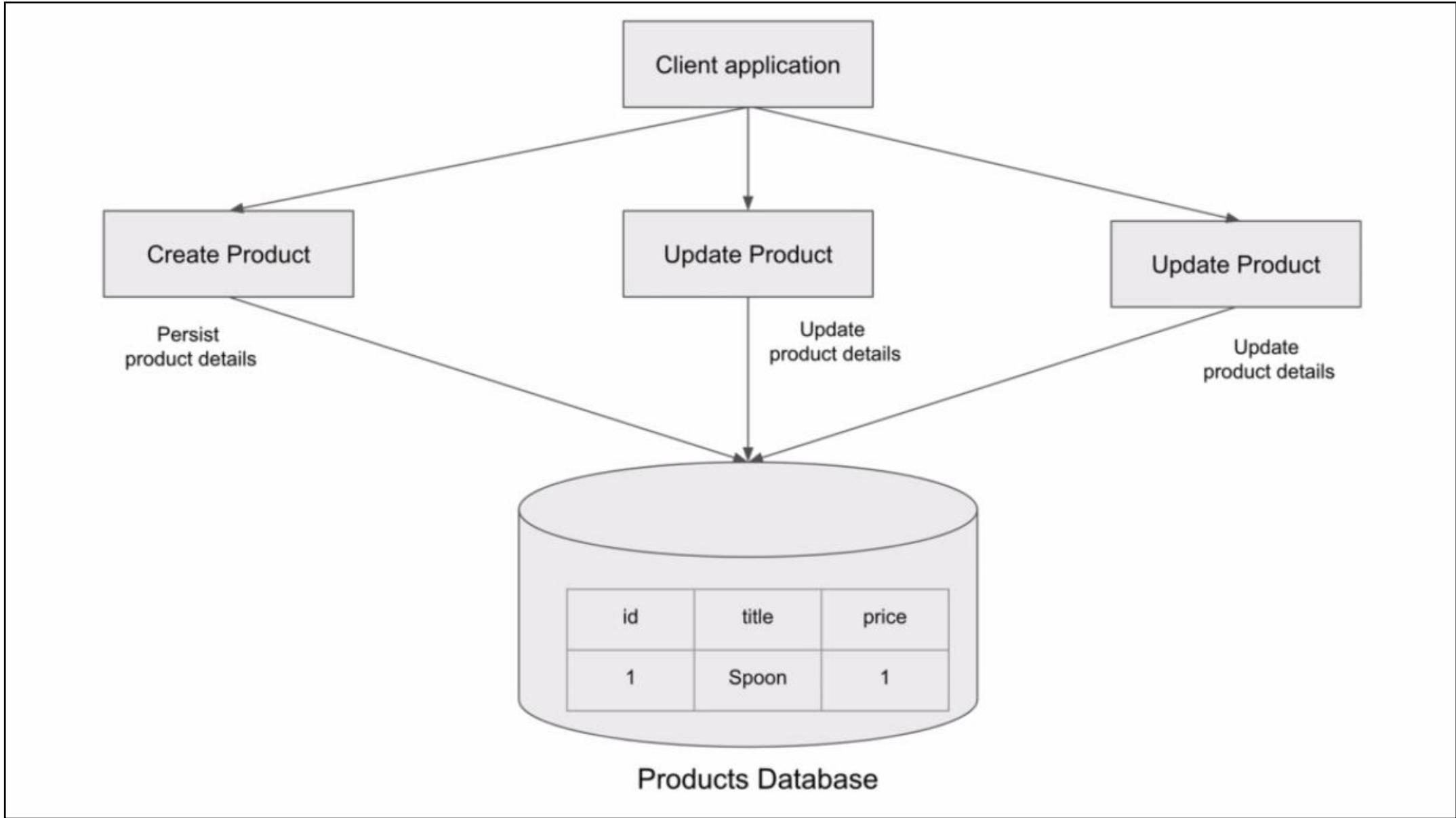


Day - 1

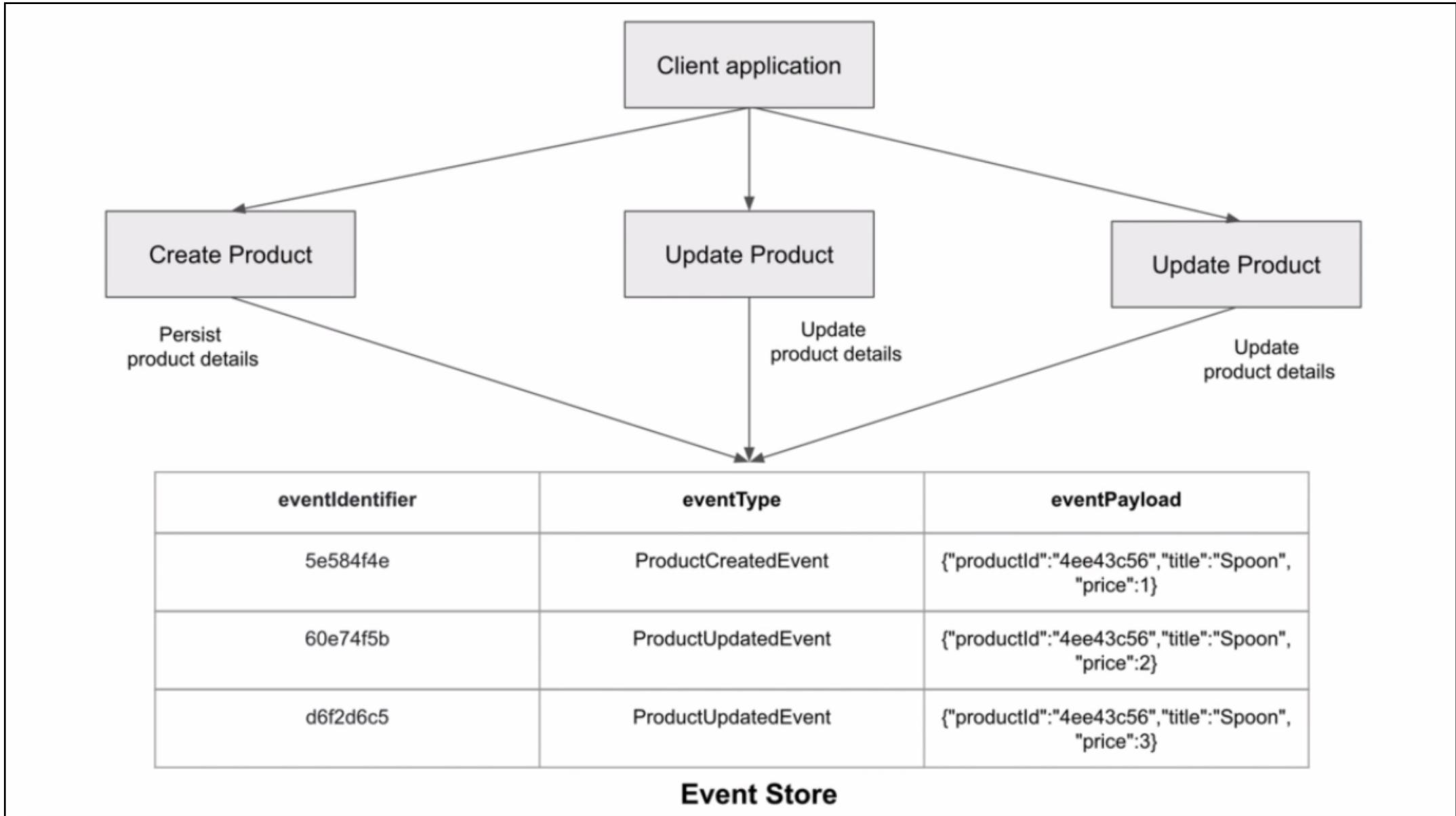
CQRS and Event Sourcing

- 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.

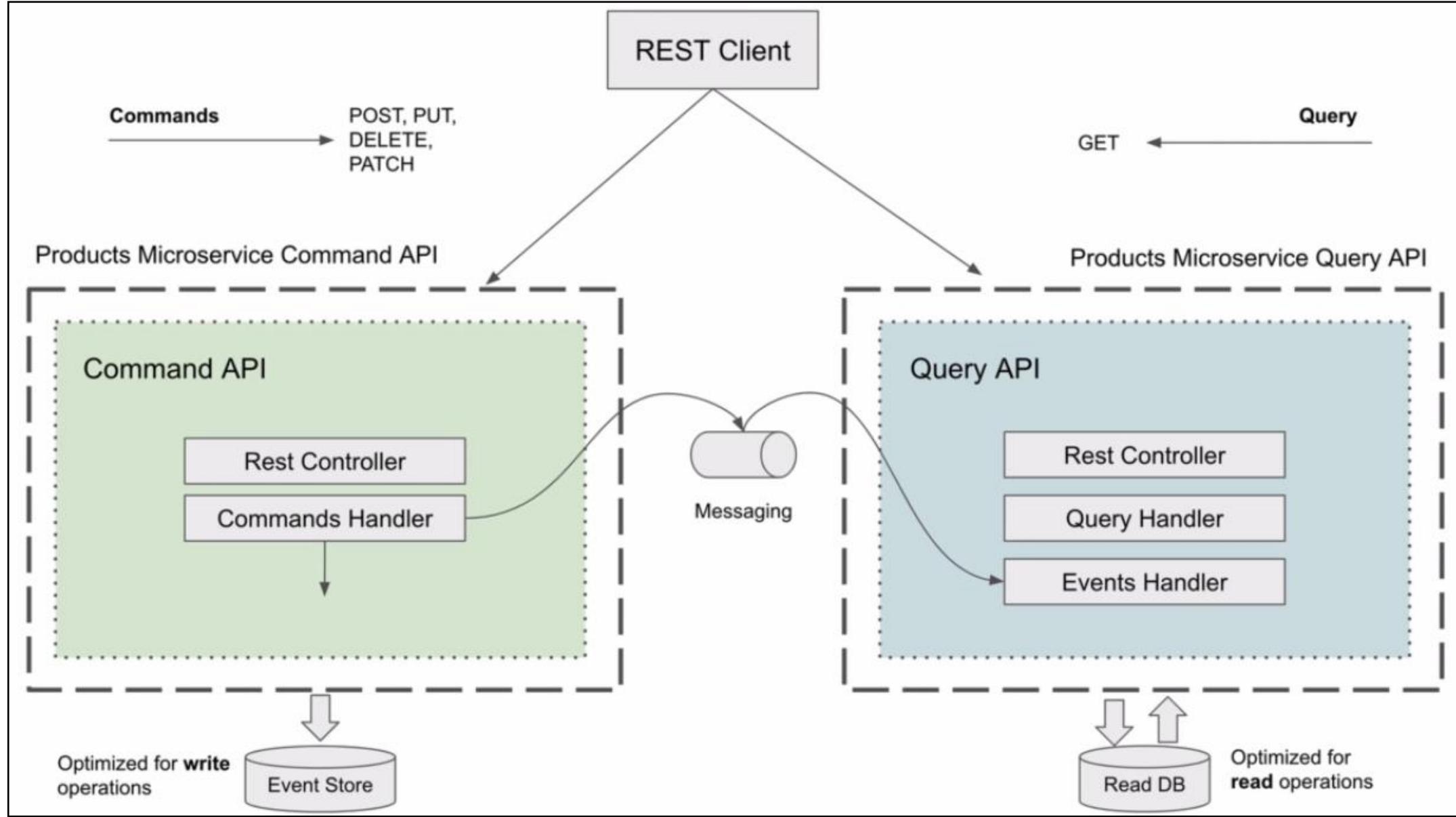
Event Sourcing



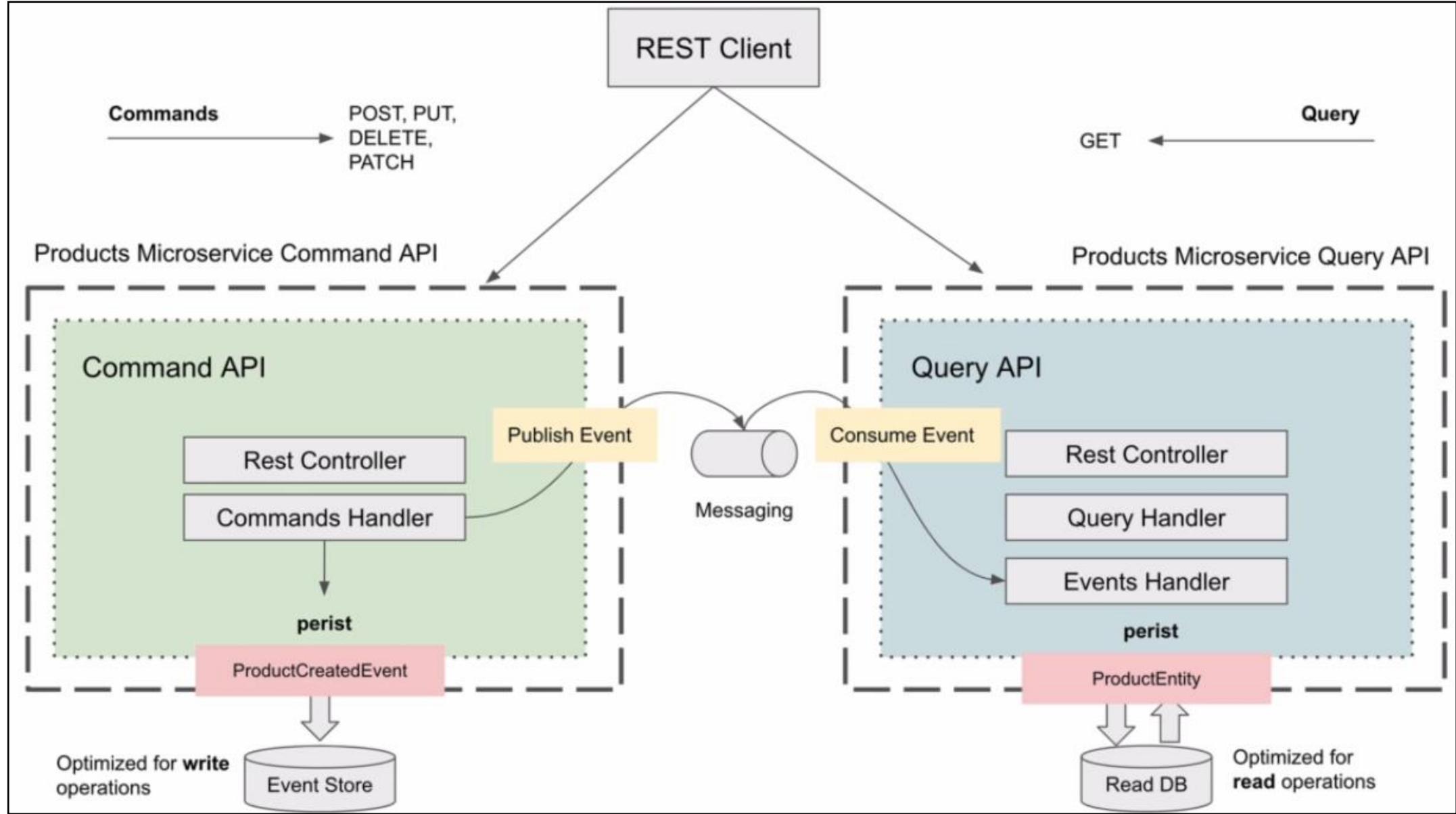
Event Sourcing



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 expose 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
1
2server.port=8761
3eureka.client.register-with-eureka=false
4eureka.client.fetch-registry=false
5eureka.instance.prefer-ip-address=true
6#eureka.instance.hostname=localhost
7eureka.client.service-url.defaultZone=http://localhost:8761/eureka
8
```

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
2eureka.client.service-url.defaultZone=http://localhost:8761/eureka
3spring.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

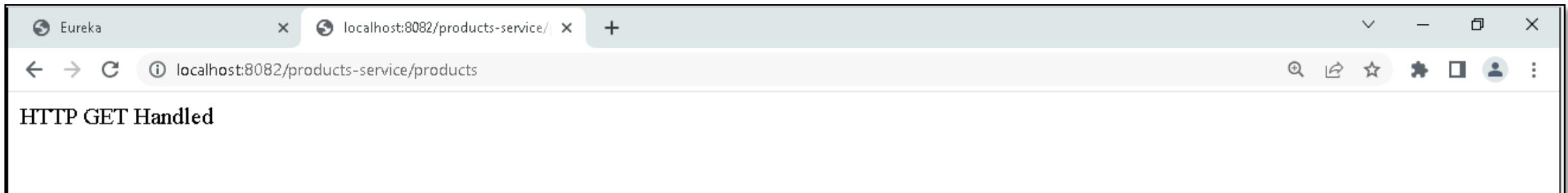
1. Ensure the Discovery Server and Product Service is running.
2. Now let's implement an API gateway that acts as a single-entry point for a collection of microservices.
3. Create a new Project for **ApiGateway** using the Spring Initializr (start.spring.io).
4. Select the Spring Boot Version - 2.7.13.
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
2spring.application.name=api-gateway
3server.port=8082
4eureka.client.service-url.defaultZone=http://localhost:8761/eureka
5
6spring.cloud.gateway.discovery.locator.enabled=true
7spring.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>



- 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 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.



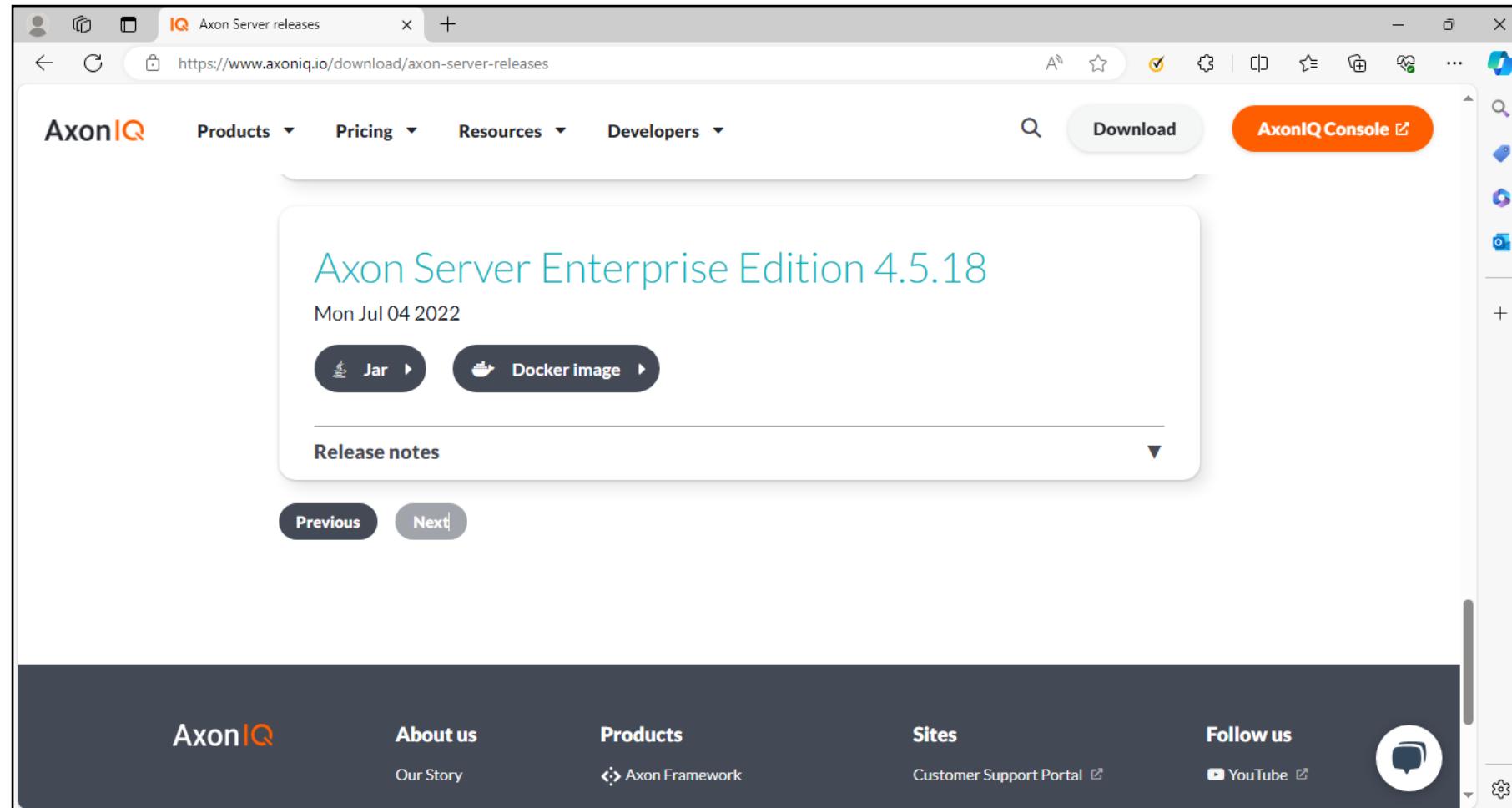
Day - 2

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>

The screenshot shows the Axon Server dashboard running at <http://localhost:8024>. The left sidebar has icons for Settings, Overview, Search, Commands, and Queries, with 'Commands' currently selected. The main area displays the following information:

Configuration	Status	License
Node Name: microservices	Last event token: -1	Edition: Standard Edition
Host Name: microservices	Activity in the last minute:	
Http Port: 8024	Commands: 0	
GRPC Port: 8124		

Two yellow message bars are present: 'SSL disabled' and 'Authentication disabled'.



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
```

The screenshot shows a Windows Command Prompt window titled "Command Prompt". The command entered is "docker run -d --name axonserver -p 8024:8024 -p 8124:8124 axoniq/axonserver:4.5.8". The output of the command is displayed, showing the progress of pulling the image from the repository. It lists several layers being pulled, their IDs, and the final digest. After the pull is complete, it shows the status as "Downloaded newer image for axoniq/axonserver:4.5.8". The container ID is listed as "a2006525b087". Finally, the "docker ps" command is run to list the running containers, showing the "axonserver" container with its details.

```
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
5347aaaf66df0: 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:

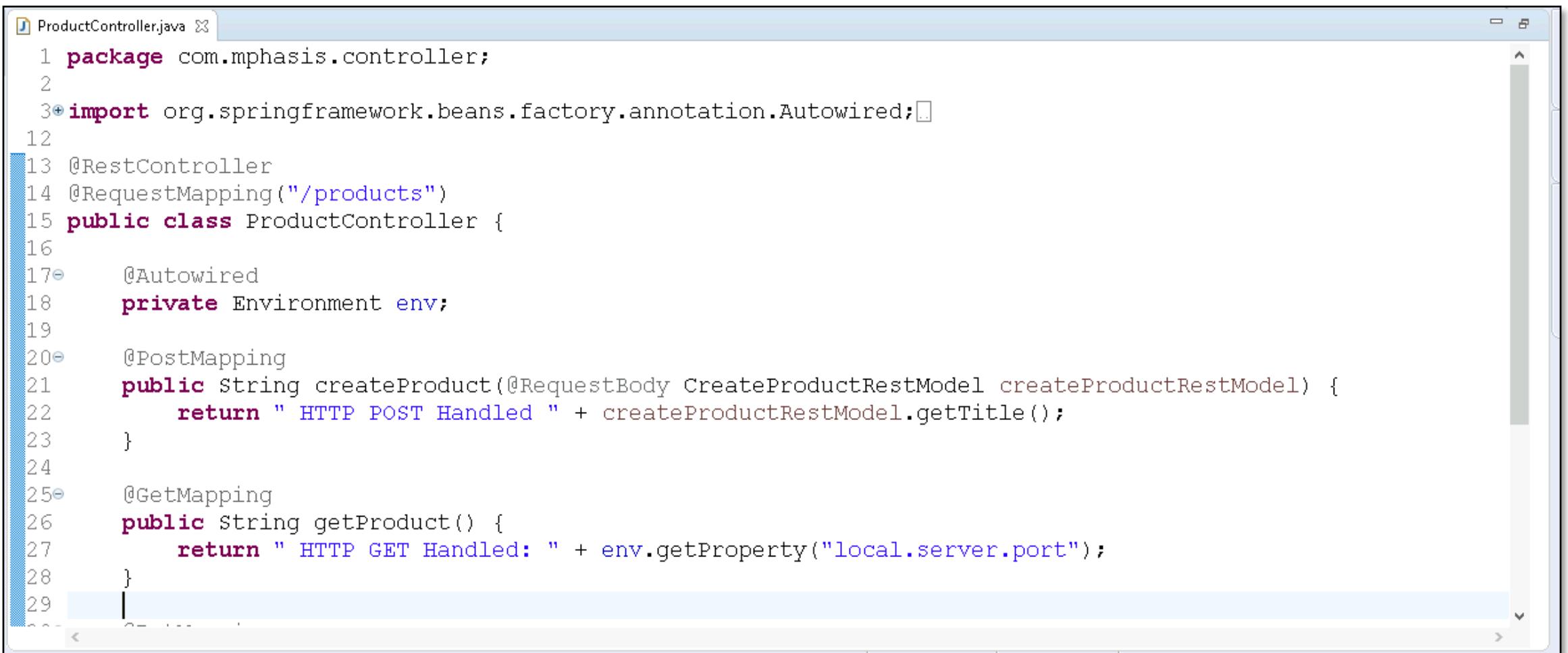
The screenshot shows a Java code editor window with the title bar "CreateProductRestModel.java". The code is as follows:

```
1 package com.mphasis.controller;
2
3 import java.math.BigDecimal;
4
5 import lombok.Data;
6
7 @Data
8 public class CreateProductRestModel {
9
10     private String title;
11     private BigDecimal price;
12     private Integer quantity;
13 }
14
```



Accept HTTP Request Body

- Apply **CreateProductRestModel** to the Controller:



The screenshot shows a Java code editor window with the file `ProductController.java` open. The code defines a REST controller for products. It includes imports for `com.mphasis.controller`, `org.springframework.beans.factory.annotation.Autowired`, and `org.springframework.web.bind.annotation.RestController`, `@RequestMapping`, `@PostMapping`, and `@GetMapping`. The controller has two methods: `createProduct` which handles POST requests and returns a message with the product title, and `getProduct` which handles GET requests and returns a message with the local server port.

```
ProductController.java
1 package com.mphasis.controller;
2
3+import org.springframework.beans.factory.annotation.Autowired;□
12
13 @RestController
14 @RequestMapping("/products")
15 public class ProductController {
16
17@    @Autowired
18    private Environment env;
19
20@    @PostMapping
21    public String createProduct(@RequestBody CreateProductRestModel createProductRestModel) {
22        return " HTTP POST Handled " + createProductRestModel.getTitle();
23    }
24
25@    @GetMapping
26    public String getProduct() {
27        return " HTTP GET Handled: " + env.getProperty("local.server.port");
28    }
29}
```



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



TargetAggregateIdentifier

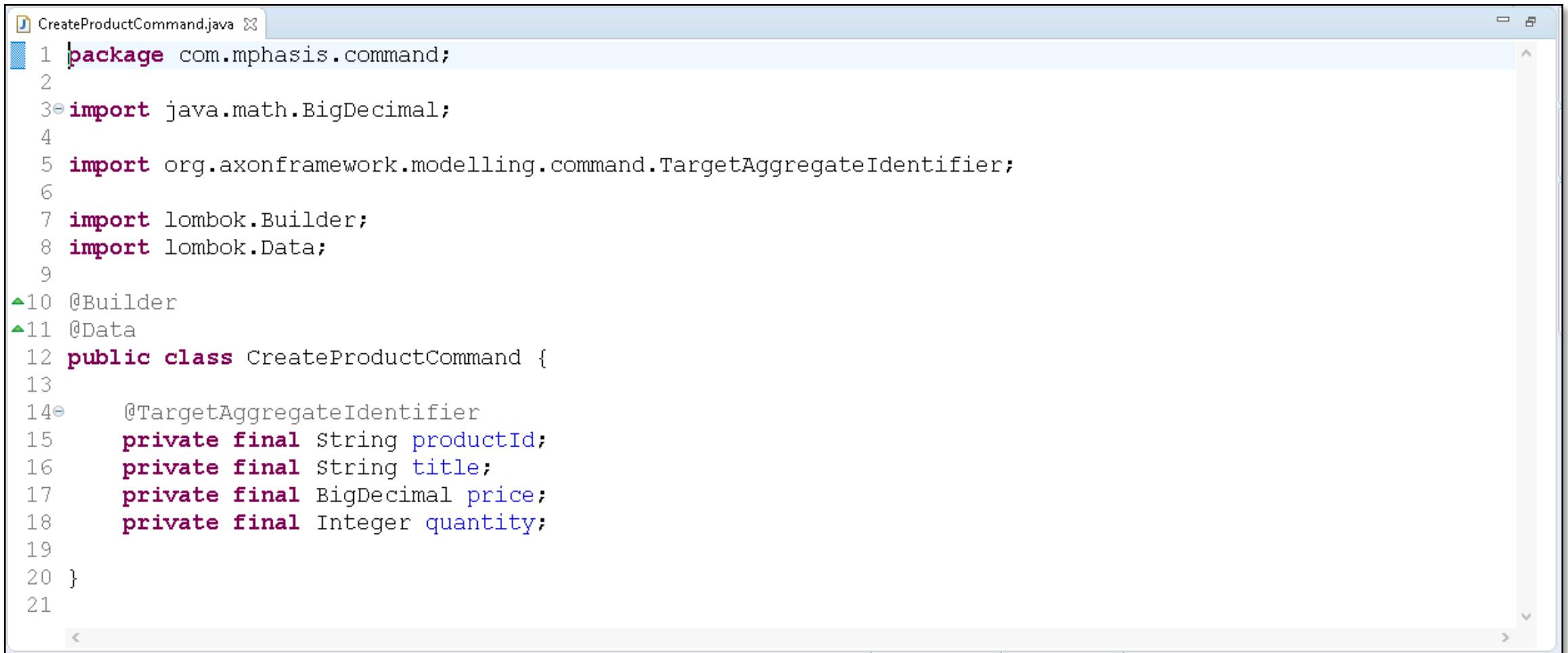


The `TargetAggregateIdentifier` annotation tells Axon that the annotated field is an id of a given aggregate to which the command should be targeted.



Creating a new Command class

- The CreateProductCommand will be read-only:



```
1 package com.mphasis.command;
2
3 import java.math.BigDecimal;
4
5 import org.axonframework.modelling.command.TargetAggregateIdentifier;
6
7 import lombok.Builder;
8 import lombok.Data;
9
10 @Builder
11 @Data
12 public class CreateProductCommand {
13
14     @TargetAggregateIdentifier
15     private final String productId;
16     private final String title;
17     private final BigDecimal price;
18     private final Integer quantity;
19
20 }
```



Send Command to a Command Gateway

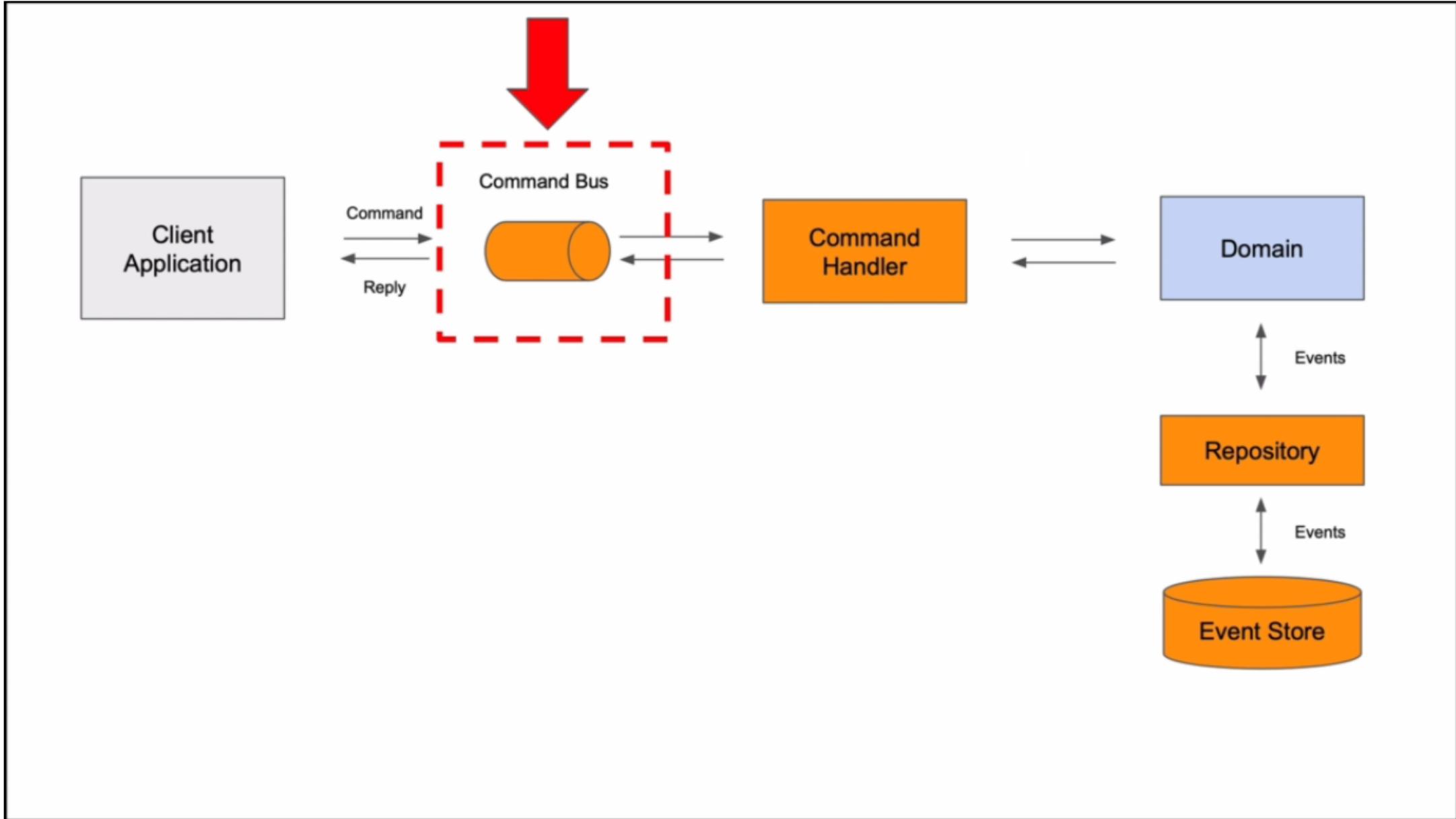


CommandGateway:



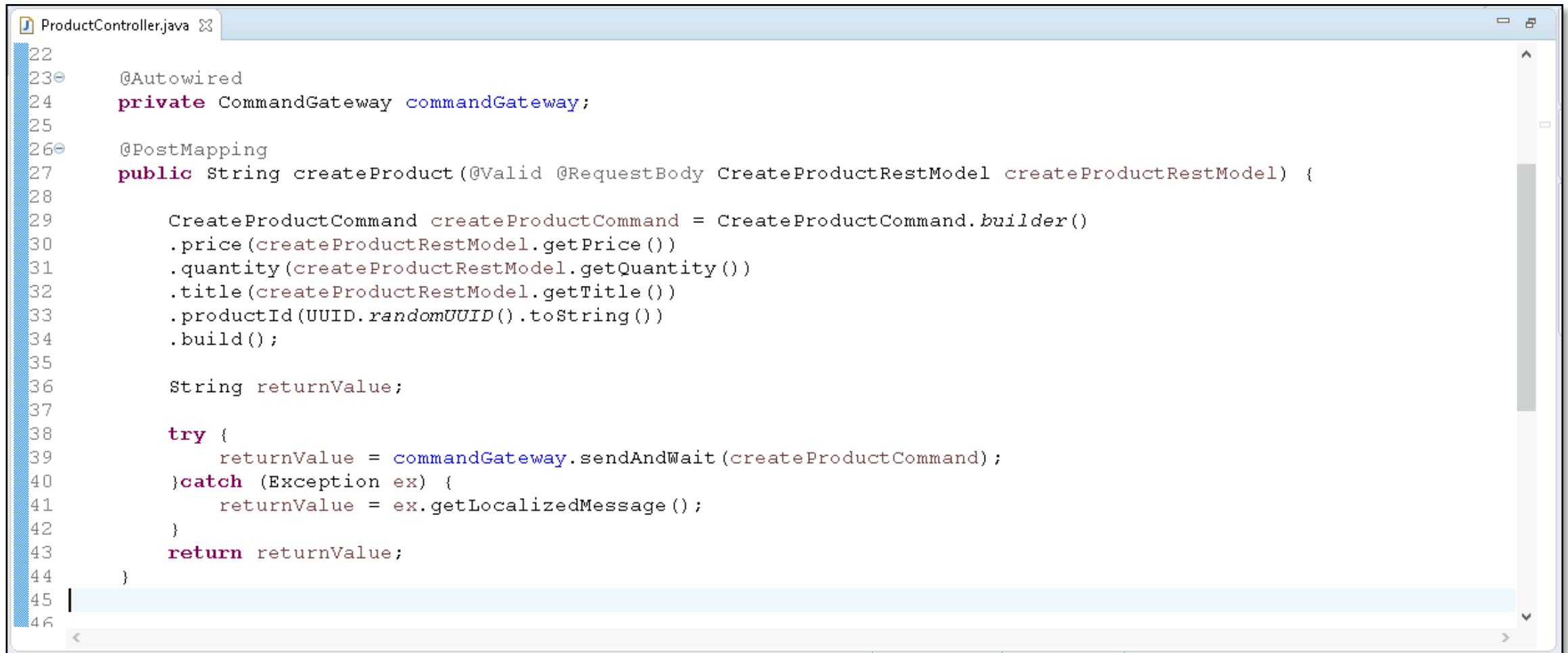
Interface towards the Command Handling components of an application. This interface provides a friendlier API toward the command bus. The CommandGateway allows for components dispatching commands to wait for the result.

Send Command to a Command Gateway





Send Command to a Command Gateway



The screenshot shows a Java code editor window with the file `ProductController.java` open. The code implements a `CreateProduct` command using a builder pattern and sends it to a `CommandGateway`.

```
22
23@Autowired
24 private CommandGateway commandGateway;
25
26@PostMapping
27 public String createProduct(@Valid @RequestBody CreateProductRestModel createProductRestModel) {
28
29     CreateProductCommand createProductCommand = CreateProductCommand.builder()
30         .price(createProductRestModel.getPrice())
31         .quantity(createProductRestModel.getQuantity())
32         .title(createProductRestModel.getTitle())
33         .productId(UUID.randomUUID().toString())
34         .build();
35
36     String returnValue;
37
38     try {
39         returnValue = commandGateway.sendAndWait(createProductCommand);
40     }catch (Exception ex) {
41         returnValue = ex.getLocalizedMessage();
42     }
43     return returnValue;
44 }
45
46
```



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

Product**Created**Event

Product**Shipped**Event

Product**Deleted**Event



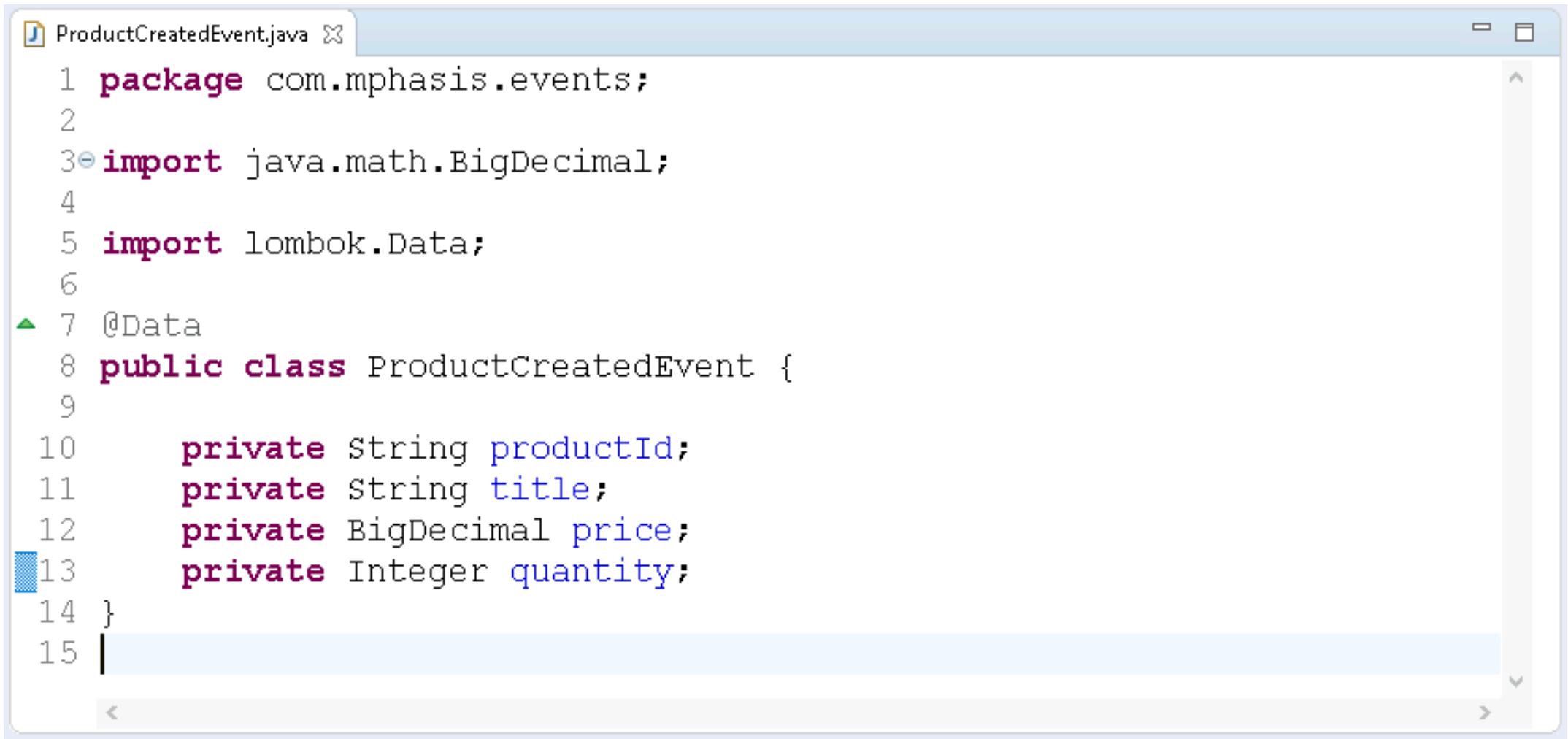
Creating a new Event class

- 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:



The screenshot shows a Java code editor window with the file `ProductCreatedEvent.java` open. The code defines a class `ProductCreatedEvent` with private fields for product ID, title, price, and quantity, annotated with `@Data` from the `lombok` library.

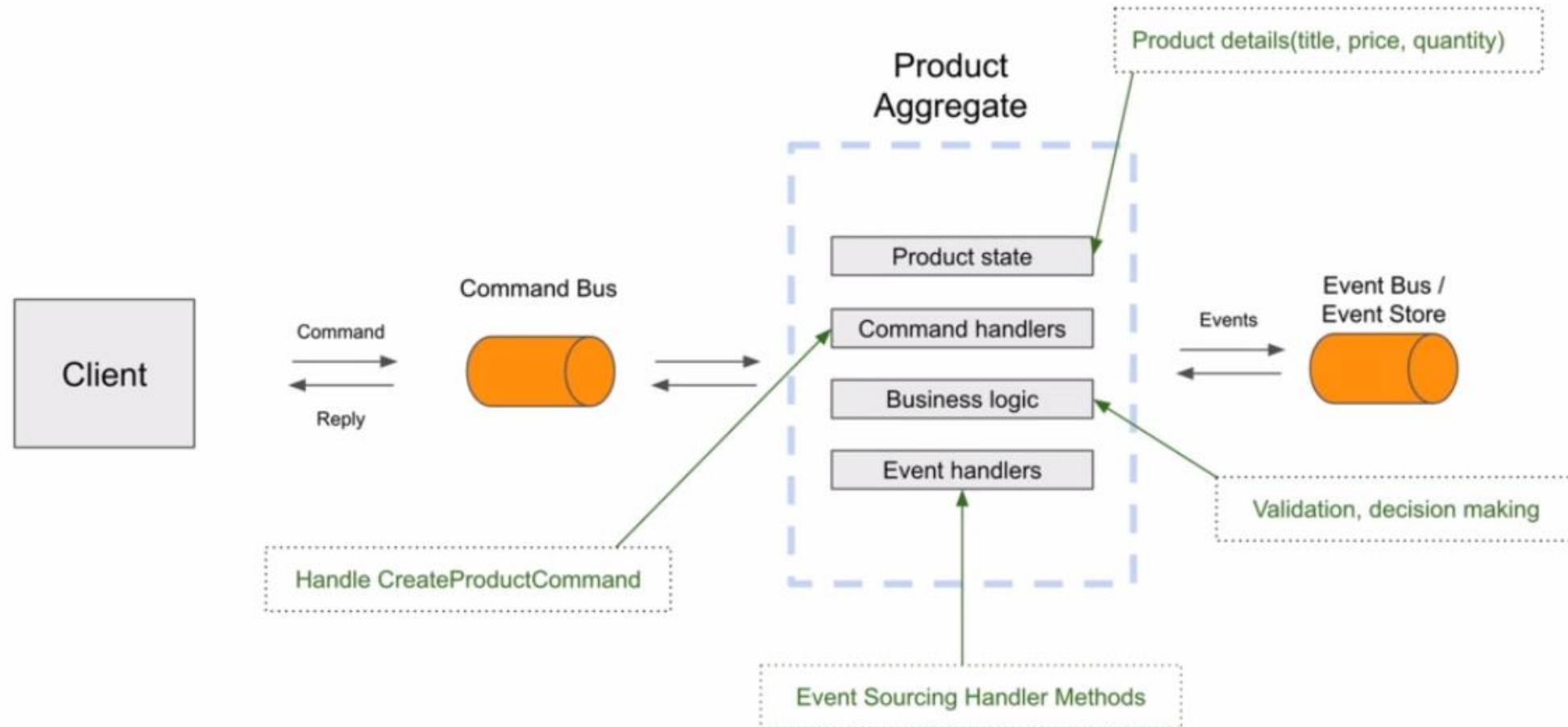
```
ProductCreatedEvent.java
1 package com.mphasis.events;
2
3 import java.math.BigDecimal;
4
5 import lombok.Data;
6
7 @Data
8 public class ProductCreatedEvent {
9
10    private String productId;
11    private String title;
12    private BigDecimal price;
13    private Integer quantity;
14 }
15 |
```



Day - 2

The Command Model – Product Aggregate

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



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 X
1 package com.mphasis.command;
2
3 import org.axonframework.commandhandling.CommandHandler;
4 import org.axonframework.spring.stereotype.Aggregate;
5
6 @Aggregate
7 public class ProductAggregate {
8
9     public ProductAggregate() {
10
11
12 }
13
14 @CommandHandler
15 public ProductAggregate(CreateProductCommand createProductCommand) {
16     // Validate Create Product Command
17 }
18 }
```



Validate the CreateProductCommand

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



```
ProductAggregate.java X
8 @Aggregate
9 public class ProductAggregate {
10
11     public ProductAggregate() {
12
13     }
14
15     @CommandHandler
16     public ProductAggregate(CreateProductCommand createProductCommand) {
17         // Validate Create Product Command
18
19         if (createProductCommand.getPrice().compareTo(BigDecimal.ZERO) <= 0) {
20             throw new IllegalArgumentException("Price cannot be less or equal than zero");
21         }
22
23         if (createProductCommand.getTitle() == null
24             || createProductCommand.getTitle().isEmpty()) {
25             throw new IllegalArgumentException("Title cannot be empty");
26         }
27     }
28 }
```



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

```
18
19  @CommandHandler
20  public ProductAggregate(CreateProductCommand createProductCommand) {
21      // Validate Create Product Command
22
23      if (createProductCommand.getPrice().compareTo(BigDecimal.ZERO) <= 0) {
24          throw new IllegalArgumentException("Price cannot be less or equal than zero");
25      }
26
27      if (createProductCommand.getTitle() == null
28          || createProductCommand.getTitle().isEmpty()) {
29          throw new IllegalArgumentException("Title cannot be empty");
30      }
31
32      ProductCreatedEvent productCreatedEvent = new ProductCreatedEvent();
33
34      BeanUtils.copyProperties(createProductCommand, productCreatedEvent);
35
36      AggregateLifecycle.apply(productCreatedEvent);
37  }
38 }
```

- **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



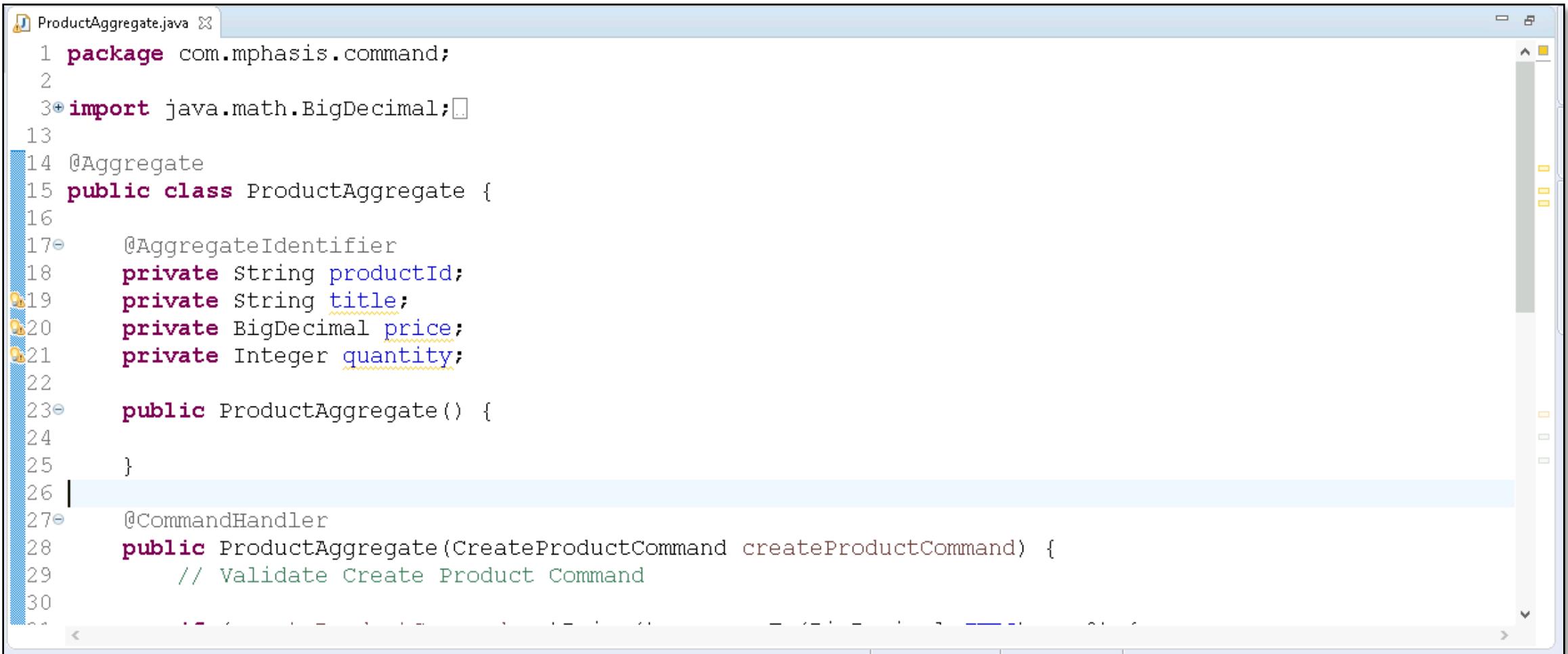
AggregateIdentifier



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



@EventSourcingHandler

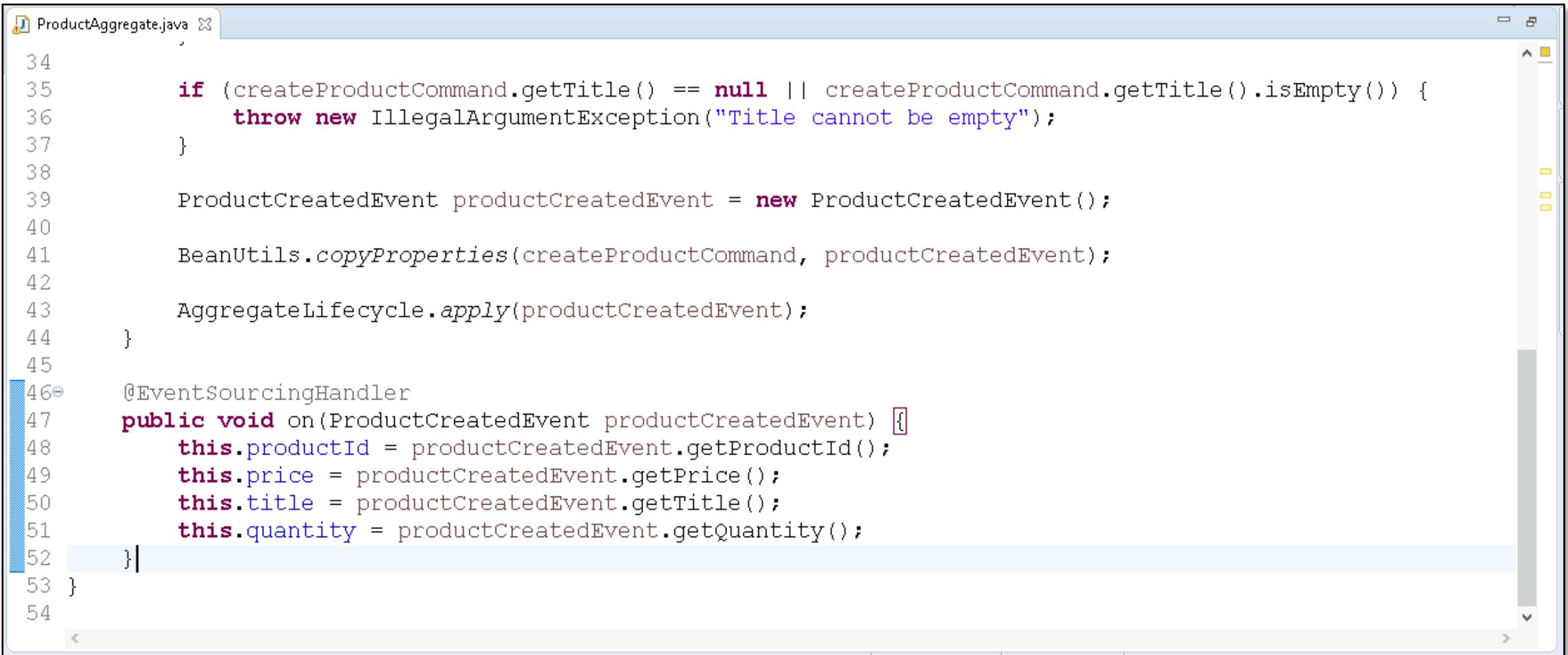


```
ProductAggregate.java ✘
1 package com.mphasis.command;
2
3+import java.math.BigDecimal;□
13
14 @Aggregate
15 public class ProductAggregate {
16
17@ AggregateIdentifier
18    private String productId;
19    private String title;
20    private BigDecimal price;
21    private Integer quantity;
22
23@ Public
24    public ProductAggregate() {
25
26    }
27@ CommandHandler
28    public ProductAggregate(CreateProductCommand createProductCommand) {
29        // Validate Create Product Command
30
31    }
32}
```

The screenshot shows a Java code editor window with the file 'ProductAggregate.java' open. The code defines a class 'ProductAggregate' with fields for product ID, title, price, and quantity. It includes a constructor and a command handler method. The code editor has syntax highlighting and a vertical scrollbar on the right.



@EventSourcingHandler



```
ProductAggregate.java X
34
35     if (createProductCommand.getTitle() == null || createProductCommand.getTitle().isEmpty()) {
36         throw new IllegalArgumentException("Title cannot be empty");
37     }
38
39     ProductCreatedEvent productCreatedEvent = new ProductCreatedEvent();
40
41     BeanUtils.copyProperties(createProductCommand, productCreatedEvent);
42
43     AggregateLifecycle.apply(productCreatedEvent);
44 }
45
46@EventSourcingHandler
47 public void on(ProductCreatedEvent productCreatedEvent) {
48     this.productId = productCreatedEvent.getProductId();
49     this.price = productCreatedEvent.getPrice();
50     this.title = productCreatedEvent.getTitle();
51     this.quantity = productCreatedEvent.getQuantity();
52 }
53 }
54
```



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

The screenshot shows the Postman application interface. At the top, there is a navigation bar with 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. Below the navigation bar, a list of requests is shown, with one selected: 'POST http://localhost:8082/products-service/products'. The request details panel shows the method 'POST' and the URL 'http://localhost:8082/products-service/products'. The 'Body' tab is selected, showing the JSON payload:

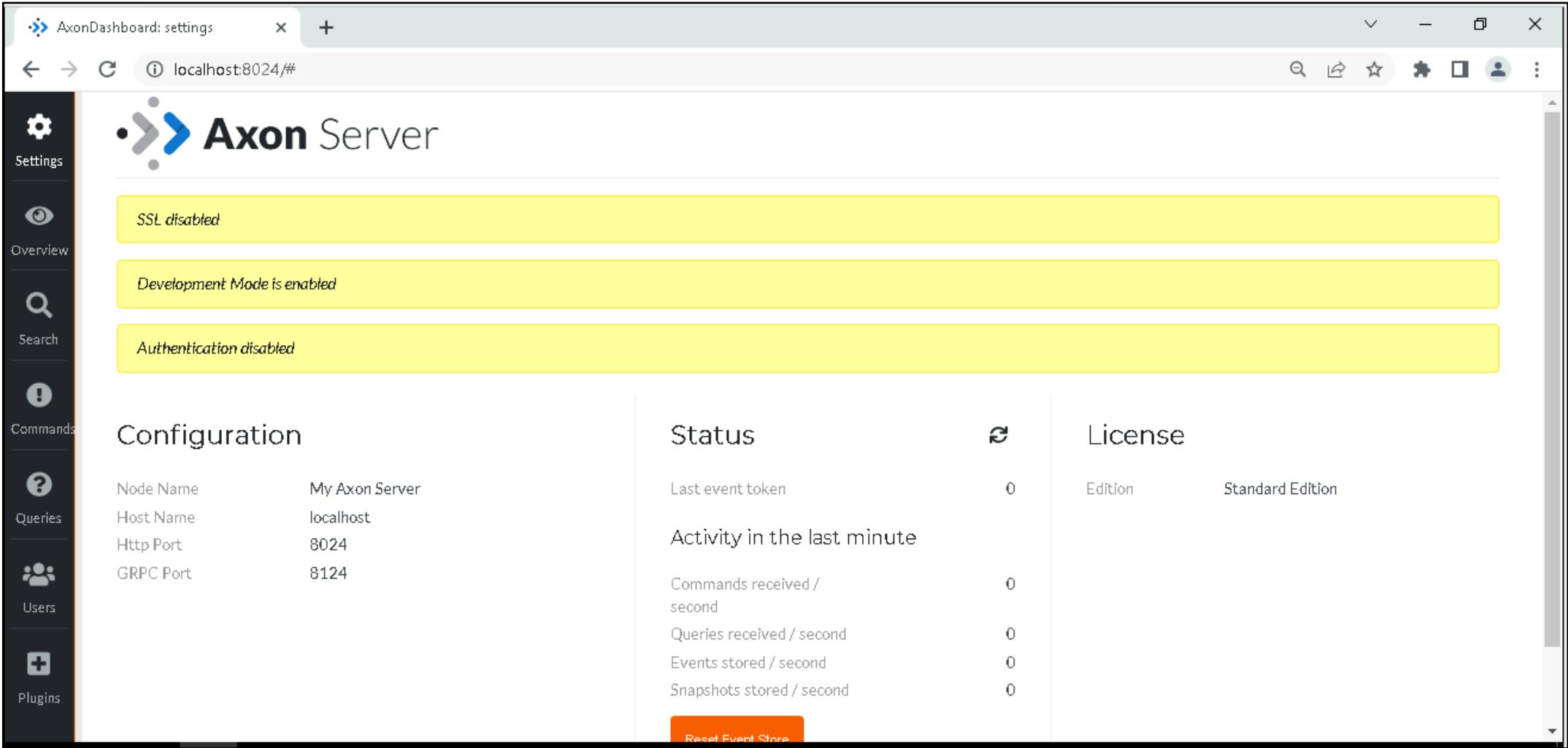
```
1 {  
2   "title": "iPhone 13",  
3   "price": 300,  
4   "quantity": 2  
5 }  
6
```

The 'Body' tab also includes tabs for 'Cookies', 'Beautify', and 'Pretty' (which is selected). Below the body, the response status is shown as 'Status: 200 OK' with a timestamp of 'Time: 1m 18.78 s' and a size of 'Size: 152 B'. The 'Save Response' button is also visible. At the bottom of the interface, there is a taskbar with various icons and a system tray showing the date and time '6:49 PM 7/4/2023'.



Previewing Event in the EventStore

- Go to Axon Server:



The screenshot shows the Axon Dashboard interface at `localhost:8024/#`. The left sidebar has icons for Settings, Overview, Search, Commands, Queries, Users, and Plugins. The main area displays the following information:

Axon Server

- SSL disabled
- Development Mode is enabled
- Authentication disabled

Configuration

Node Name	My Axon Server
Host Name	localhost
Http Port	8024
GRPC Port	8124

Status

Metric	Value
Last event token	0
Activity in the last minute	0
Commands received / second	0
Queries received / second	0
Events stored / second	0
Snapshots stored / second	0

License

Edition	Standard Edition
Edition	Standard Edition

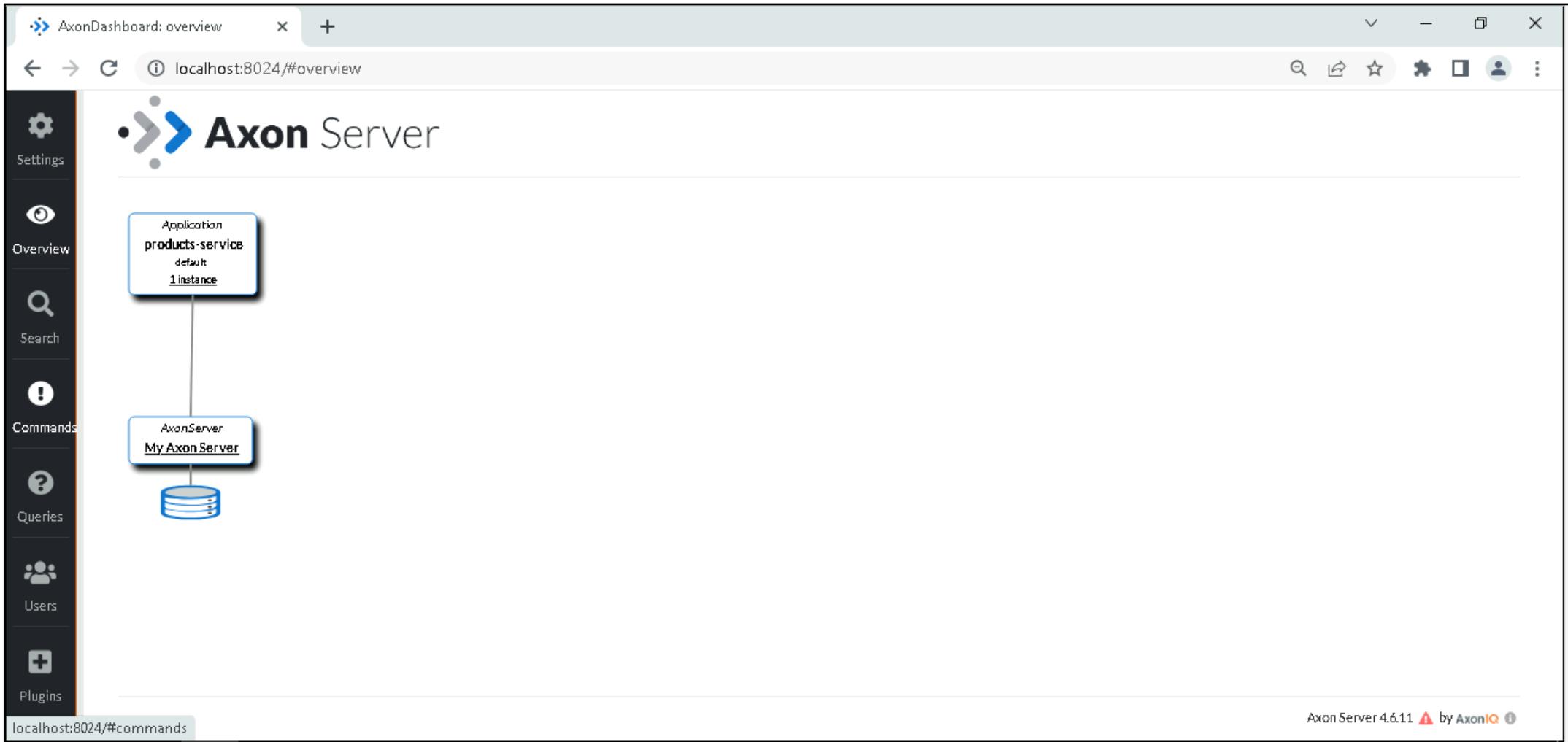
Buttons

- Reset Event Store



Previewing Event in the EventStore

- Click on **Overview** tab:





Previewing Event in the EventStore

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

The screenshot shows the Axon Dashboard interface for the 'products-service' application. The left sidebar contains navigation links: Settings, Overview, Search, Commands, Queries (selected), Users, and Plugins. The main content area displays the 'Axon Server' logo and 'Application details for products-service'. It includes a 'Subscription Queries' section with three metrics: #Total Subscription Queries (0), #Active Subscription Queries (0), and #Updates to Subscription Queries (0). Below this is a table showing an instance named '3656@microsoftservices' running on 'My Axon Server' (default node). The 'Command' section shows the command 'com.mphasis.command.CreateProductCommand'. At the bottom, there is a table with columns: Query, Response Types, #Subscriptions, #Active Subscriptions, and #Updates.

Query	Response Types	#Subscriptions	#Active Subscriptions	#Updates

Axon Server 4.6.11 by AxonIQ



Previewing Event in the EventStore

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

The screenshot shows the Axon Server dashboard interface. On the left, there is a vertical sidebar with icons for Settings, Overview, Search (which is selected), Commands, Queries, Users, and Plugins. The main area has a title 'Axon Server' with a logo. It includes a search bar with 'Events' selected, a query time window dropdown set to 'last hour', and a 'Live Updates' checkbox. Below the search bar is a text input field 'Enter your query here' and a large orange 'Search' button. A section titled 'About the query language' is present. The main content area displays a table of event data with the following columns: token, eventIdentifier, aggregateIdentifier, aggregateType, payloadType, payloadData, timestamp, and metaData. One row of data is shown, corresponding to the event listed in the search bar. At the bottom right, there are buttons for 'Rows per page' (set to 10), navigation (1-1 of 1), and pagination controls.

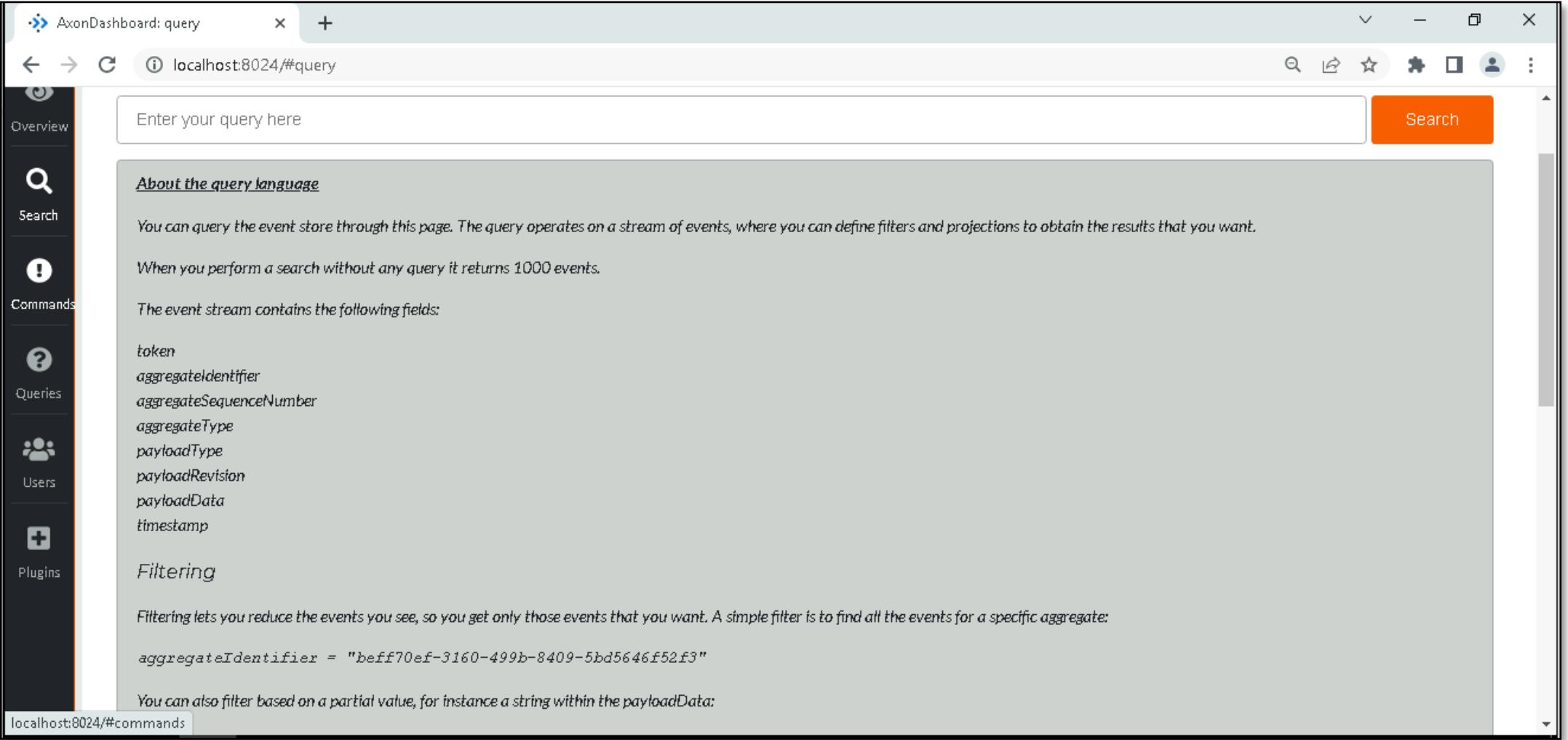
token	eventIdentifier	aggregateIdentifier	aggregateType	payloadType	payloadData	timestamp	metaData	
0	f6ce3e9c-3fe...	51a8bedc-b3...	0	ProductAggr...	com.mphasis.events.Product...	<com.mphasis.events.ProductCreatedEve...	2023-07-04...	{traceId=c...

Axon Server 4.6.11 by AxonIQ



Previewing Event in the EventStore

- About the **query** language:



The screenshot shows the AxonDashboard: query interface running on localhost:8024. The left sidebar has a vertical navigation menu with icons and labels: Overview (selected), Search, Commands, Queries, Users, and Plugins. The main content area displays the 'About the query language' page. It includes a search bar at the top with the placeholder 'Enter your query here' and an orange 'Search' button. Below the search bar, there's a section titled 'About the query language' with the following text:
You can query the event store through this page. The query operates on a stream of events, where you can define filters and projections to obtain the results that you want.
When you perform a search without any query it returns 1000 events.
The event stream contains the following fields:
`token`
`aggregateIdentifier`
`aggregateSequenceNumber`
`aggregateType`
`payloadType`
`payloadRevision`
`payloadData`
`timestamp`
A section titled 'Filtering' follows, with the text:
Filtering lets you reduce the events you see, so you get only those events that you want. A simple filter is to find all the events for a specific aggregate:
`aggregateIdentifier = "beff70ef-3160-499b-8409-5bd5646f52f3"`
And finally, a note about partial filtering:
You can also filter based on a partial value, for instance a string within the payloadData:



Previewing Event in the EventStore

- Search using the **aggregateIdentifier**:

The screenshot shows the Axon Dashboard interface. On the left is a sidebar with icons for Settings, Overview, Search (selected), Commands, Queries, Users, and Plugins. The main area has a title 'Axon Server' with a search bar below it. The search bar contains the query 'aggregateIdentifier = "51a8bedc-b3ca-436c-b44e-64b70d0750d3"'. Below the search bar is a section titled 'About the query language'. A table is displayed with the following data:

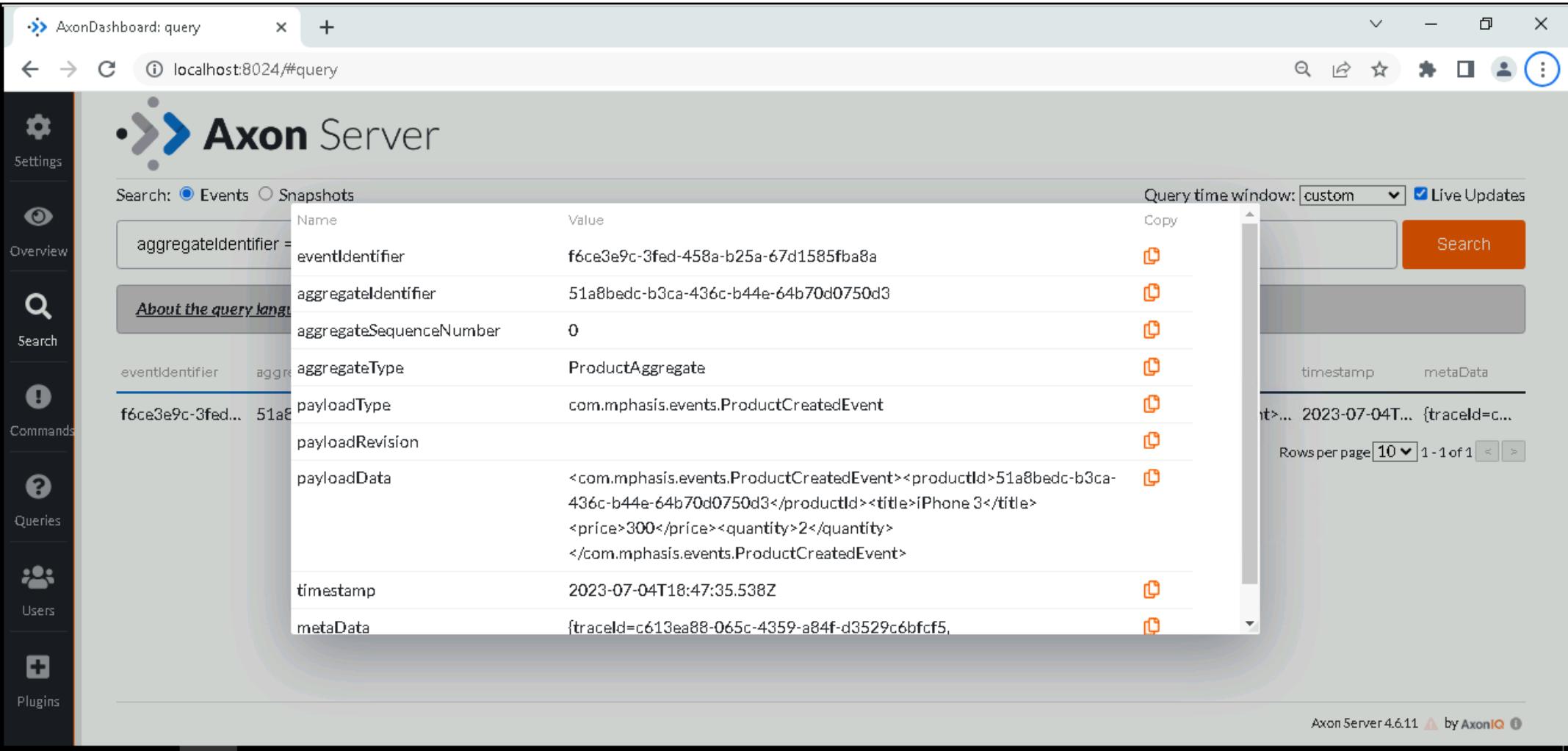
eventIdentifier	aggregateIdentifier	aggregateType	payloadType	payloadRaw	payloadData	timestamp	metaData
f6ce3e9c-3fed...	51a8bedc-b3ca...	0	ProductAggre...	com.mphasis.events.ProductCr...	<com.mphasis.events.ProductCreatedEvent>...	2023-07-04T...	{traceId=c...

At the bottom, there is a 'Rows per page' dropdown set to 10, and a footer indicating 'Axon Server 4.6.11'.



Previewing Event in the EventStore

- View the event details available in Event Store:



The screenshot shows the Axon Server dashboard at `localhost:8024/#query`. The left sidebar has a 'Queries' tab selected. The main area displays a table of event details:

Name	Value	Actions
eventIdentifier	f6ce3e9c-3fed-458a-b25a-67d1585fba8a	Copy
aggregateIdentifier	51a8bedc-b3ca-436c-b44e-64b70d0750d3	Copy
aggregateSequenceNumber	0	Copy
aggregateType	ProductAggregate	Copy
payloadType	com.mphasis.events.ProductCreatedEvent	Copy
payloadRevision		Copy
payloadData	<com.mphasis.events.ProductCreatedEvent><productId>51a8bedc-b3ca-436c-b44e-64b70d0750d3</productId><title>iPhone 3</title><price>300</price><quantity>2</quantity></com.mphasis.events.ProductCreatedEvent>	Copy
timestamp	2023-07-04T18:47:35.538Z	Copy
metaData	{traceId=c613ea88-065c-4359-a84f-d3529c6bfcf5,	Copy

Query time window: custom Live Updates

Rows per page: 10 | 1 - 1 of 1

Axon Server 4.6.11 by AxonIQ



Previewing Event in the EventStore

- Go to Commands tab:

The screenshot shows the Axon Dashboard interface with the title "AxonDashboard: commands" and the URL "localhost:8024/#commands". The left sidebar has a vertical navigation menu with icons and labels: Settings (gear), Overview (eye), Search (magnifying glass), Commands (exclamation mark), Queries (question mark), Users (people), and Plugins (plus sign). The "Commands" icon is highlighted. The main content area displays the "Axon Server" logo and the heading "Commands". Below it, there is a table with one row:

Command Type	Count
com.mphasis.command.CreateProductCommand	1

Next to the command type, the details "products-service@default" and "3656@microservices" are shown. At the bottom of the dashboard, the URL "localhost:8024/#commands" is repeated, along with the footer "Axon Server 4.6.11 ▲ by AxonIQ ⓘ".

- 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



Day - 3

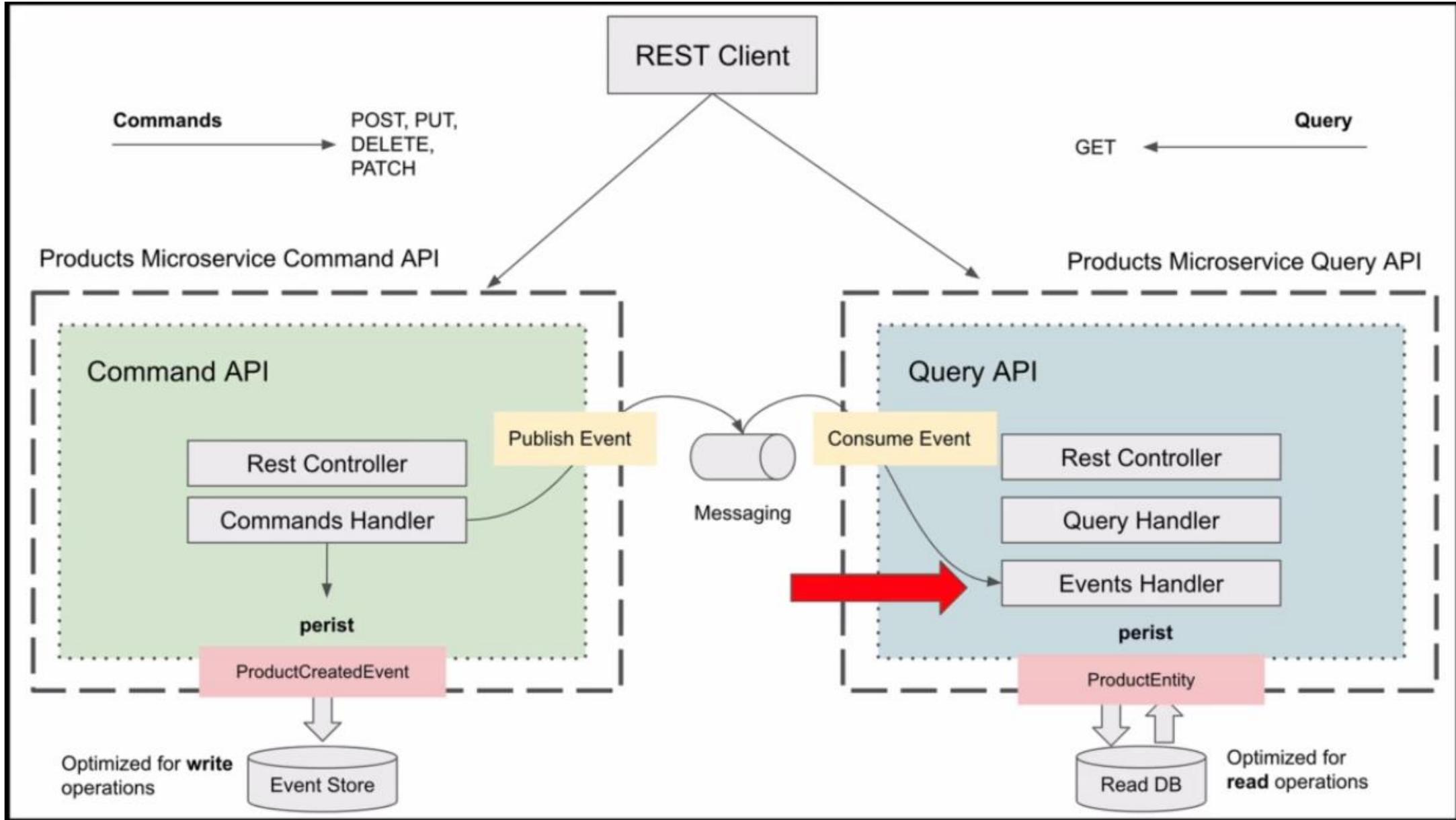
- CQRS Persisting Event in the database
- Adding Spring Data JPA & H2 dependencies
- Configure database access in the application.properties file
- Creating the Events Handler/Projection
- Implementing @EventHandler method
- CQRS, Querying Data
- Refactor Command API Rest Controller
- Create a Controller class for Query API
- Get Products Web Service Endpoint
- Implementing @QueryHandler method



Day - 3

CQRS Persisting Event in the Product Database

CQRS Persisting Event in the Product Database





Adding Spring Data JPA & H2 dependencies

- Add H2 & Spring Data JPA Starters in ProductService/pom.xml.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```



Configuring database access in application.properties file

- Add the DB properties in application.properties.

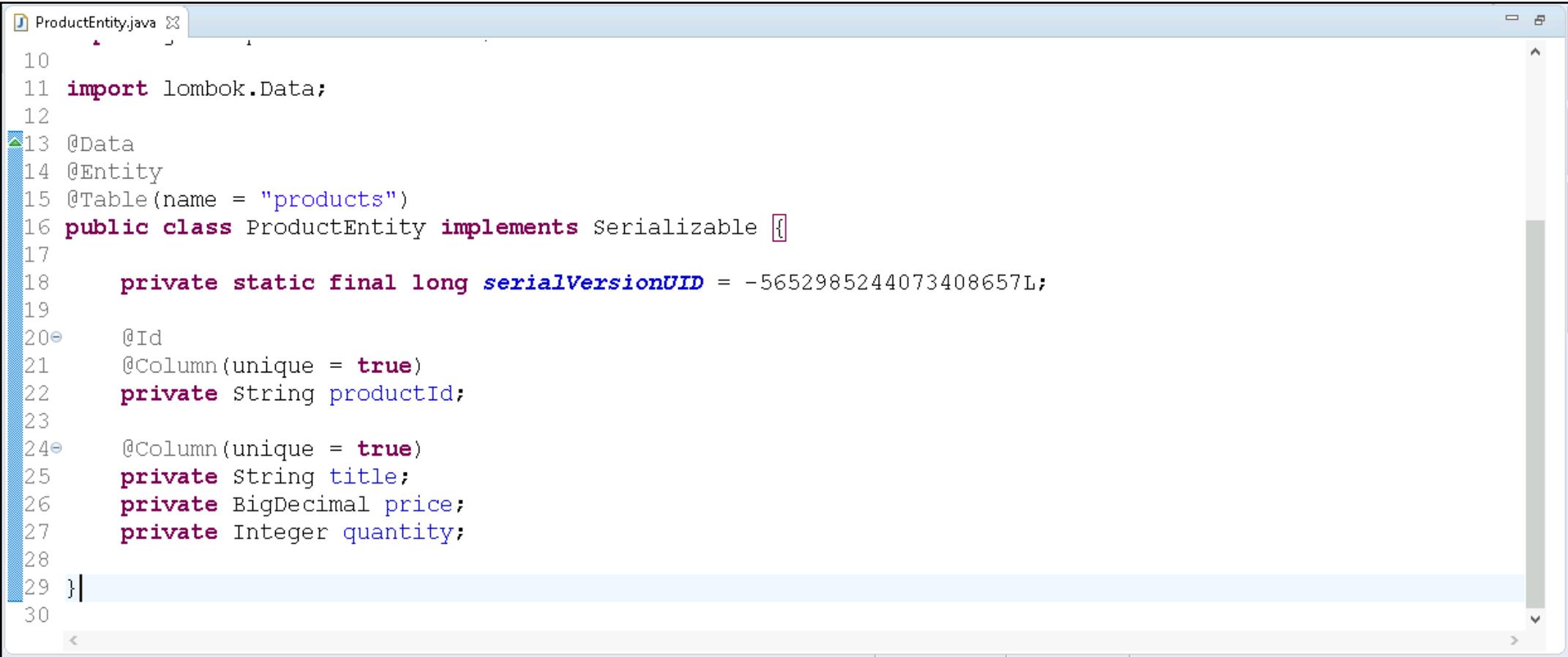
The screenshot shows a code editor window with the title bar "application.properties". The file contains the following configuration properties:

```
2 server.port=0
3 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
4 spring.application.name=products-service
5 eureka.instance.instance-id=${spring.application.name}:${instanceId:${random.value}}
6
7 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
8
9 spring.h2.console.settings.web-allow-others=true
10
11 spring.datasource.url=jdbc:h2:mem:mphasisdb
12 spring.datasource.driver-class-name=org.h2.Driver
13 spring.datasource.username=sa
14 spring.datasource.password=password
15
16 #Accessing the H2 Console
17 spring.h2.console.enabled=true
18 spring.h2.console.path=/h2-console
19
20 spring.jpa.properties.hibernate.show_sql=true
21 spring.jpa.properties.hibernate.format_sql=true
22|
```



Create a Product Entity class

- Create a ProductEntity Class:



```
ProductEntity.java
10
11 import lombok.Data;
12
13 @Data
14 @Entity
15 @Table(name = "products")
16 public class ProductEntity implements Serializable {
17
18     private static final long serialVersionUID = -5652985244073408657L;
19
20     @Id
21     @Column(unique = true)
22     private String productId;
23
24     @Column(unique = true)
25     private String title;
26     private BigDecimal price;
27     private Integer quantity;
28
29 }
```



Create a Product Repository Interface

- Create a ProductRepository Interface:

The screenshot shows a Java code editor window with the title bar "ProductRepository.java". The code is as follows:

```
1 package com.mphasis.core.data;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface ProductRepository extends JpaRepository<ProductEntity, String>{
6
7     ProductEntity findByProductId(String productId);
8     ProductEntity findByProductIdOrTitle(String productId, String title);
9 }
10
```



EventHandler



EventHandler

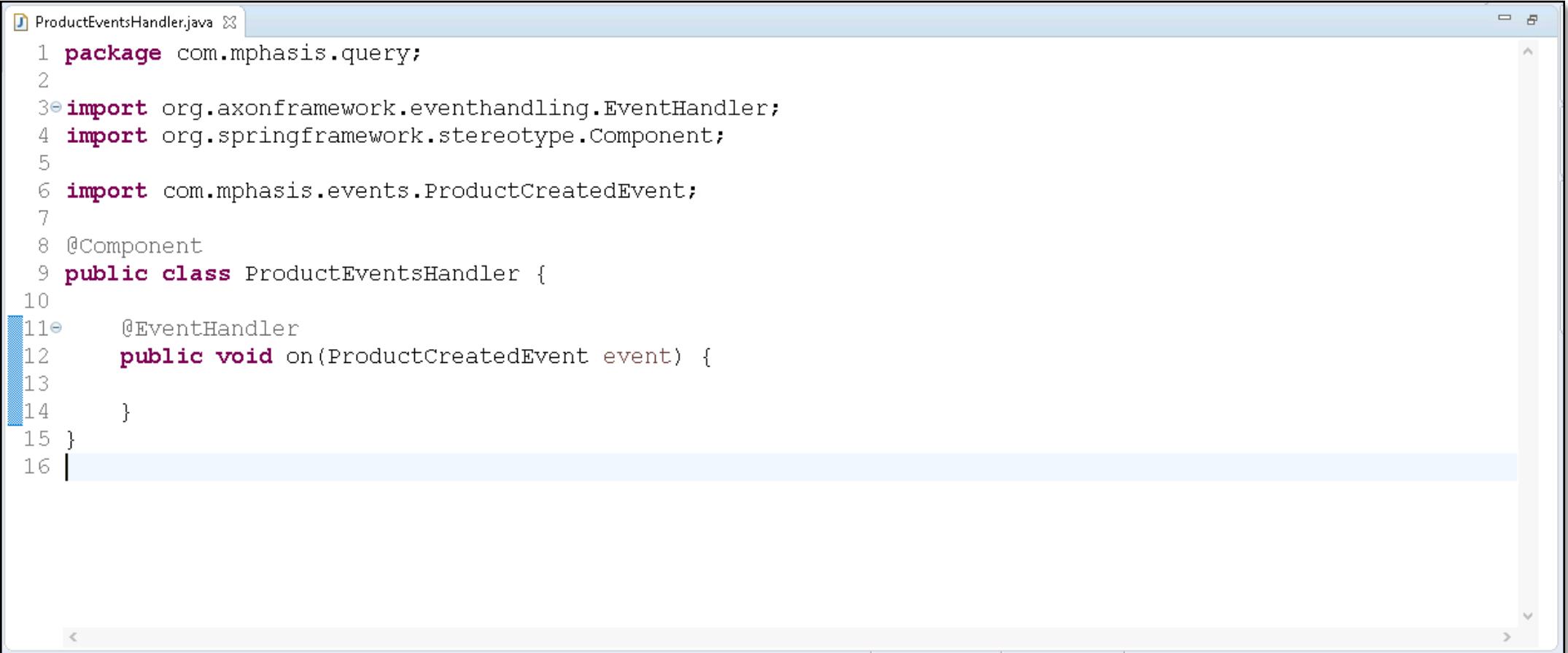


Annotation to be placed on methods that can handle events. The parameters of the annotated method are resolved using parameter resolvers.



Create a Product Events Handler Class

- Create a new ProductEventsHandler class:



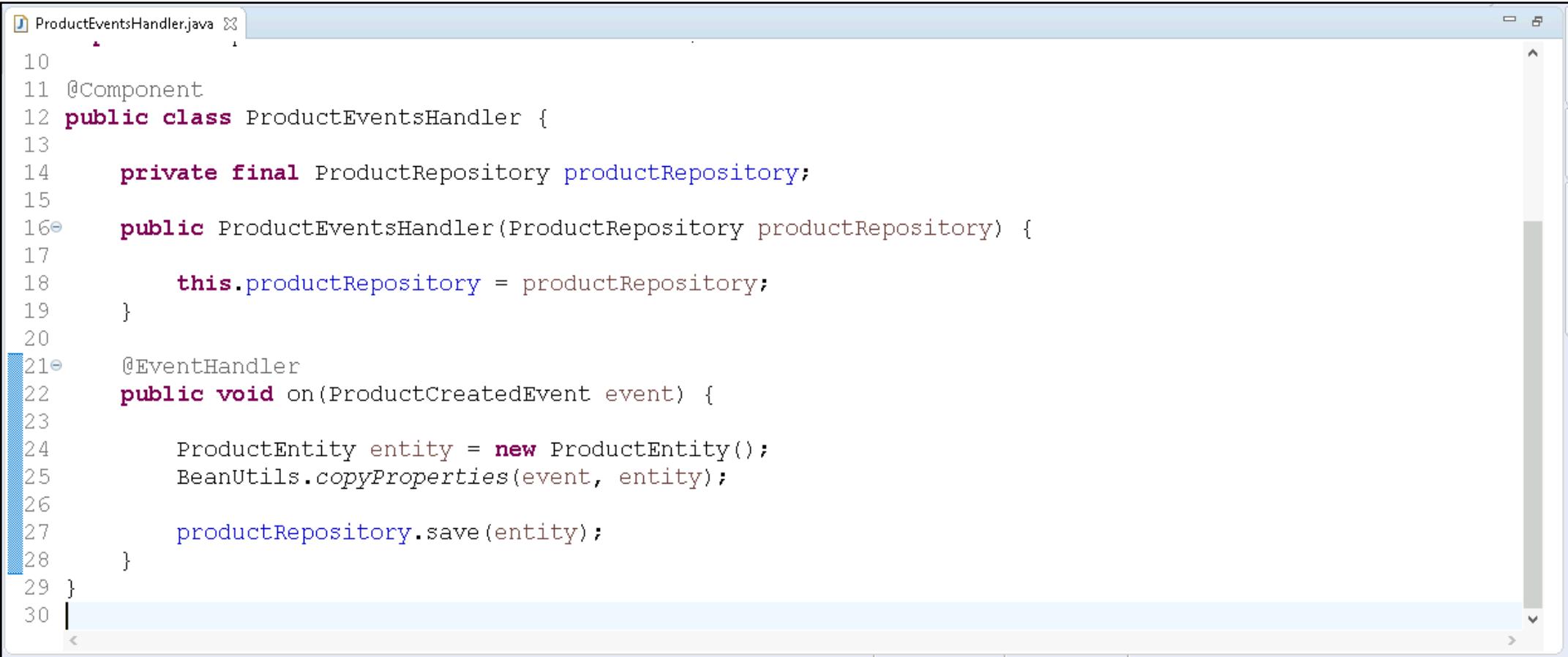
The screenshot shows a Java code editor window with the file 'ProductEventsHandler.java' open. The code defines a class 'ProductEventsHandler' that implements the 'EventHandler' interface from the Axon framework. The class has a single method 'on' that takes a 'ProductCreatedEvent' as a parameter. The code is annotated with line numbers from 1 to 16.

```
1 package com.mphasis.query;
2
3 import org.axonframework.eventhandling.EventHandler;
4 import org.springframework.stereotype.Component;
5
6 import com.mphasis.events.ProductCreatedEvent;
7
8 @Component
9 public class ProductEventsHandler {
10
11     @EventHandler
12     public void on(ProductCreatedEvent event) {
13
14     }
15 }
16
```



Implementing @EventHandler method

- Implementing @EventHandler method:



```
ProductEventsHandler.java
10
11 @Component
12 public class ProductEventsHandler {
13
14     private final ProductRepository productRepository;
15
16     public ProductEventsHandler(ProductRepository productRepository) {
17
18         this.productRepository = productRepository;
19     }
20
21     @EventHandler
22     public void on(ProductCreatedEvent event) {
23
24         ProductEntity entity = new ProductEntity();
25         BeanUtils.copyProperties(event, entity);
26
27         productRepository.save(entity);
28     }
29 }
30
```



Trying how it works

1. Add a breakpoint in the **on** method of ProductEventsHandler class.
2. Run the AxonServer using Docker command.
3. Ensure the Discovery Server (Eureka Server) is running.
4. Execute the Product Service in Debug mode.
5. Ensure the ApiGateway is running.

Let's create one more Product

The screenshot shows the Postman application interface. At the top, there is a navigation bar with 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. Below the navigation bar, a header bar shows a 'POST' method and the URL 'http://localhost:8082/products-service/products'. On the right side of this bar are buttons for 'Add to collection' and a close button. The main workspace contains a request configuration panel. The 'Method' dropdown is set to 'POST' and the 'URL' field is 'http://localhost:8082/products-service/products'. Below this, tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible, with 'Headers' currently selected. Under 'Body', the type is set to 'JSON' and the content is a JSON object:

```
1 {  
2   "title": "iPhone 1115",  
3   "price": 500,  
4   "quantity": 2  
5 }  
6
```

On the right side of the body panel, there are 'Cookies' and 'Beautify' buttons. Below the body panel, there are tabs for 'Body', 'Cookies', 'Headers (3)', and 'Test Results', with 'Body' currently selected. To the right of these tabs, status information is displayed: 'Status: 200 OK', 'Time: 548 ms', and 'Size: 152 B'. A 'Save Response' button is also present. At the bottom of the workspace, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', 'Text', and a copy icon. The bottom left corner of the workspace shows a 'Console' tab.

Preview Product record in a Database

The screenshot shows the H2 Console interface. The title bar displays three tabs: "Eureka", "H2 Console", and "AxonDashboard: query". The "H2 Console" tab is active, showing a URL: "Not secure | host.docker.internal:51578/h2-console/login.do?jsessionid=d0b63eeee1c64a142bb66a9af76e16e0". Below the tabs is a toolbar with various icons and dropdown menus for "Auto commit" (checked), "Max rows: 1000", "Auto complete" (set to Off), and "Auto select" (set to On). A sidebar on the left lists database objects: "ASSOCIATION_VALUE_ENTRY", "PRODUCTS", "SAGA_ENTRY", "TOKEN_ENTRY", "INFORMATION_SCHEMA", "Sequences", and "Users". The main area contains a SQL statement: "SELECT * FROM PRODUCTS". The results of the query are displayed in a table:

PRODUCT_ID	PRICE	QUANTITY	TITLE
51a8bedc-b3ca-436c-b44e-64b70d0750d3	300.00	2	iPhone 3
c26818f6-8b23-4497-ba1b-ba2d172367fe	500.00	2	iPhone 1115

(2 rows, 0 ms)

An "Edit" button is located below the table.



Day - 3

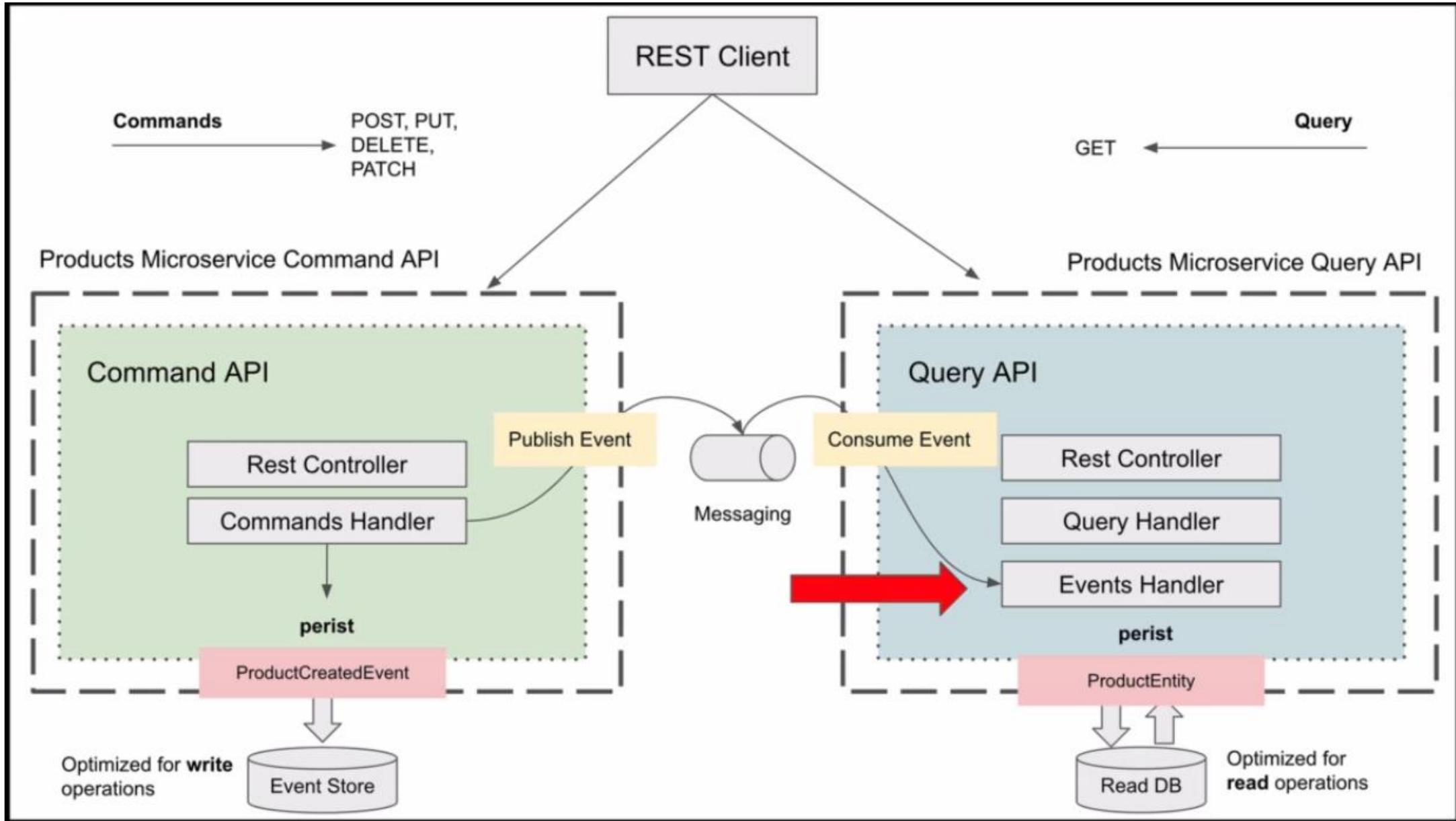
CQRS, Querying Data



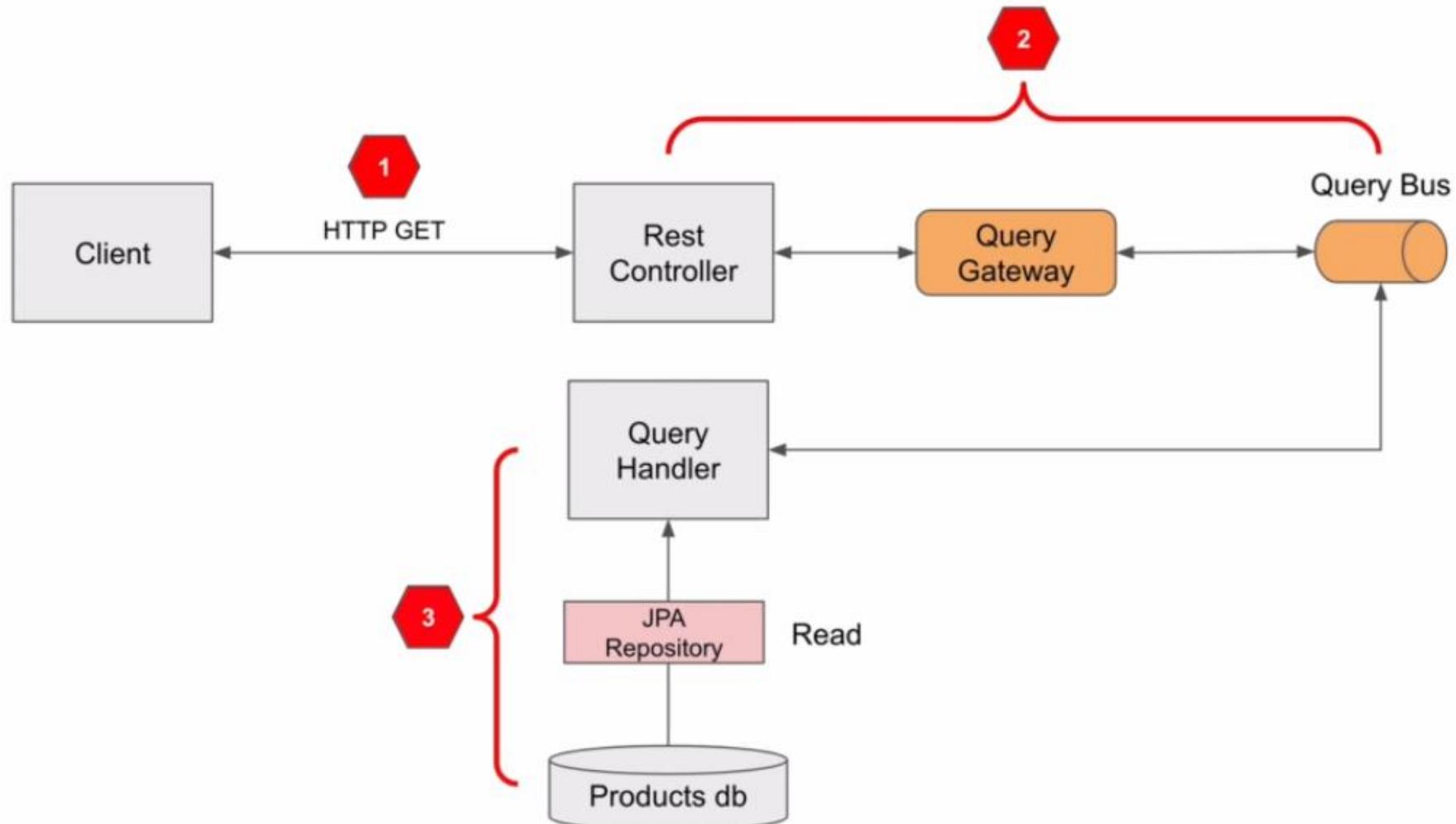
CQRS, Querying Data

- Query - express the desire for information.
- Ex: FindProductQuery, GetUserQuery

CQRS, Querying Data



CQRS, Querying Data





Create a Controller class for Query API

- Create a separate Controller class for Query API:



The screenshot shows a Java code editor window with the file 'ProductsQueryController.java' open. The code defines a REST controller for products.

```
1 package com.mphasis.query.rest;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 @RequestMapping("/products")
8 public class ProductsQueryController {
9
10 }
11
12
```



Refactor Command API Rest Controller

- Rename ProductController to ProductCommandController for better visualization.
- Rename the Package com.mphasis.controller to com.mphasis.command.rest
- Rename the Package com.mphasis.events to com.mphasis.core.events
- So, total we will have 3 main package – command, core, and query.





Get Products Web Service Endpoint

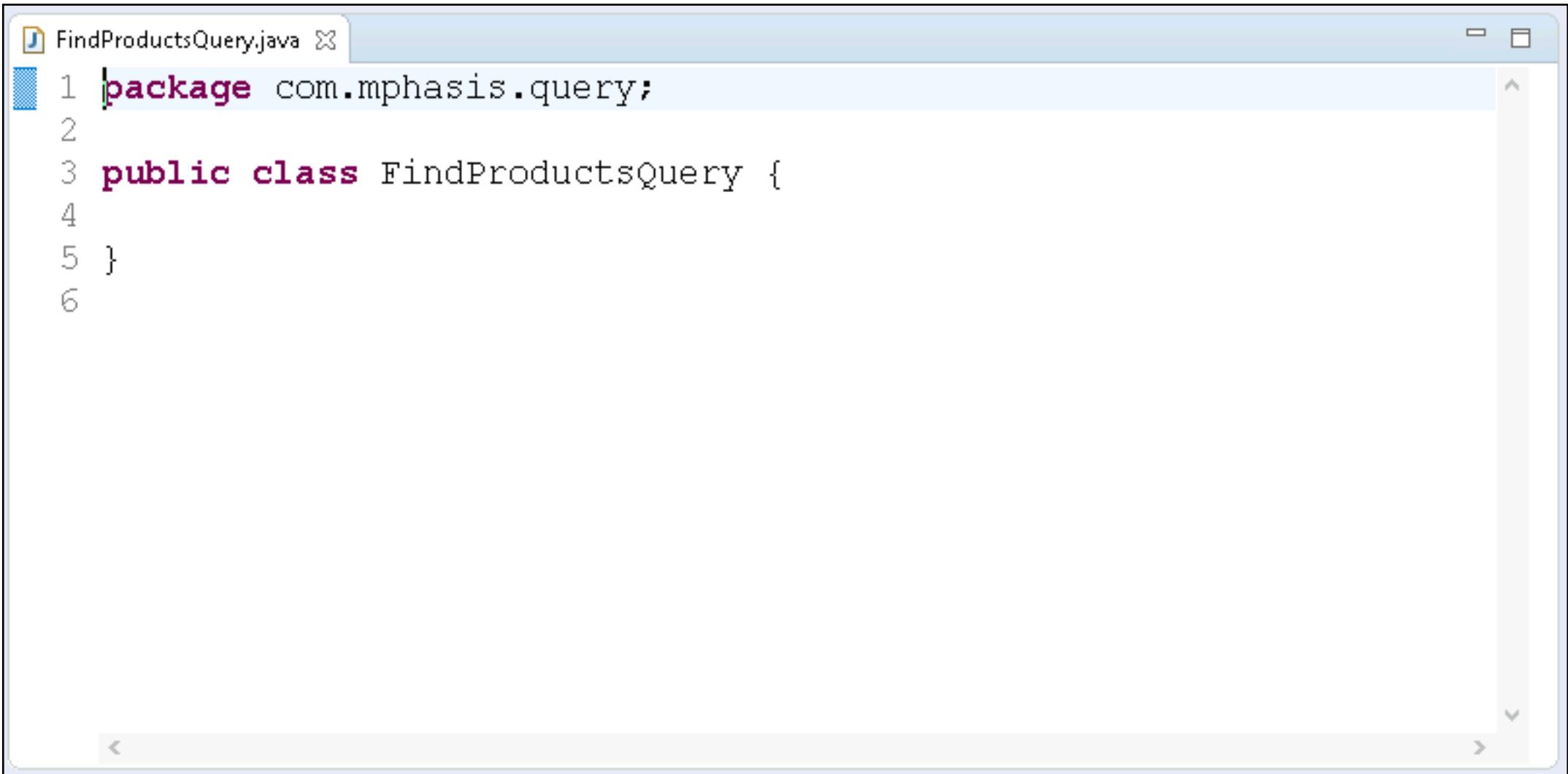
- Create a ProductRestModel class:



The screenshot shows a Java code editor window with a tab labeled "ProductRestModel.java". The code defines a class named "ProductRestModel" with private fields for product ID, title, price, and quantity, annotated with Lombok's @Data.

```
1 package com.mphasis.query.rest;
2
3 import java.math.BigDecimal;
4
5 import lombok.Data;
6
7 @Data
8 public class ProductRestModel {
9
10     private String productId;
11     private String title;
12     private BigDecimal price;
13     private Integer quantity;
14 }
15
```

- Query the QueryGateway:



The screenshot shows a Java code editor window with a single file named "FindProductsQuery.java". The code is as follows:

```
1 package com.mphasis.query;
2
3 public class FindProductsQuery {
4
5 }
6
```



QueryGateway

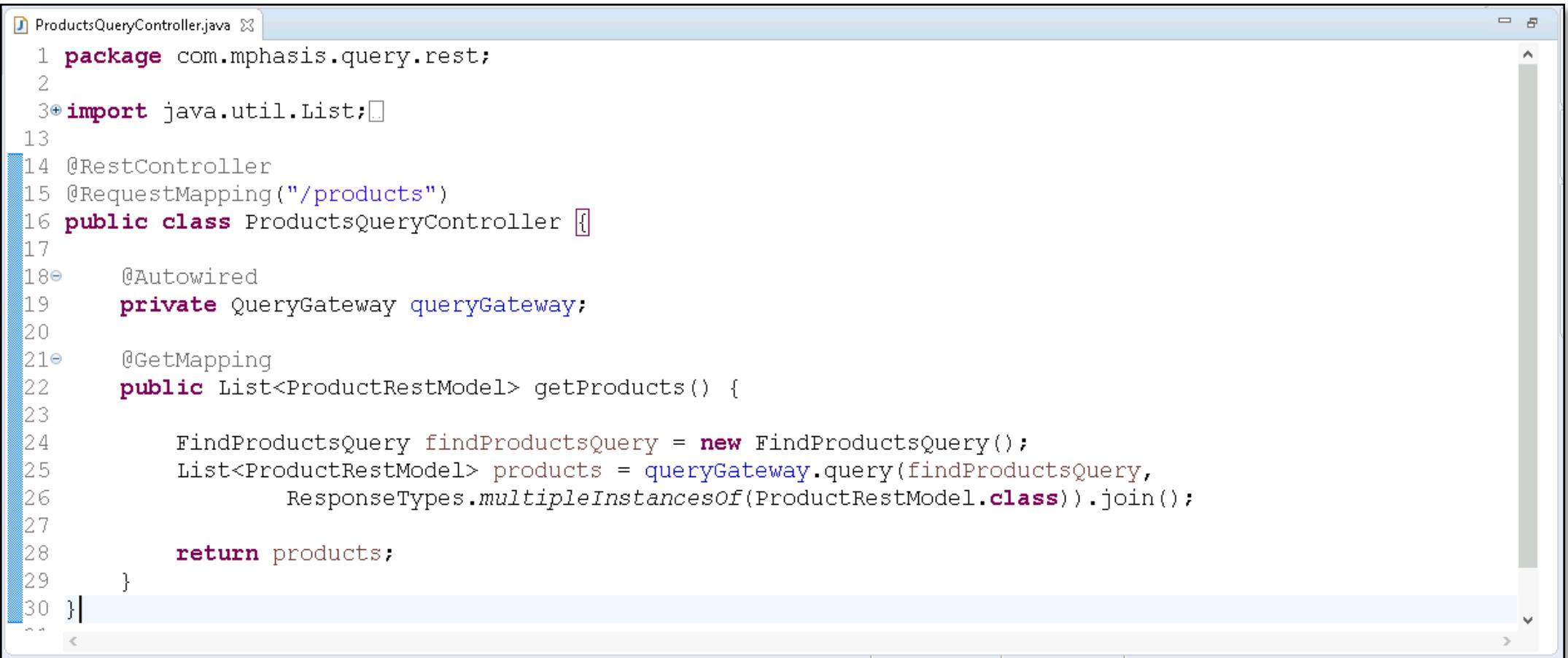


QueryGateway



Interface towards the Query Handling components of an application.
This interface provides a friendlier API toward the query bus.

- Query the QueryGateway:



```
ProductsQueryController.java
1 package com.mphasis.query.rest;
2
3+import java.util.List;
4
5 @RestController
6 @RequestMapping("/products")
7 public class ProductsQueryController {
8
9     @Autowired
10    private QueryGateway queryGateway;
11
12    @GetMapping
13    public List<ProductRestModel> getProducts() {
14
15        FindProductsQuery findProductsQuery = new FindProductsQuery();
16        List<ProductRestModel> products = queryGateway.query(findProductsQuery,
17            ResponseTypes.multipleInstancesOf(ProductRestModel.class)).join();
18
19        return products;
20    }
21}
```



QueryHandler



QueryHandler

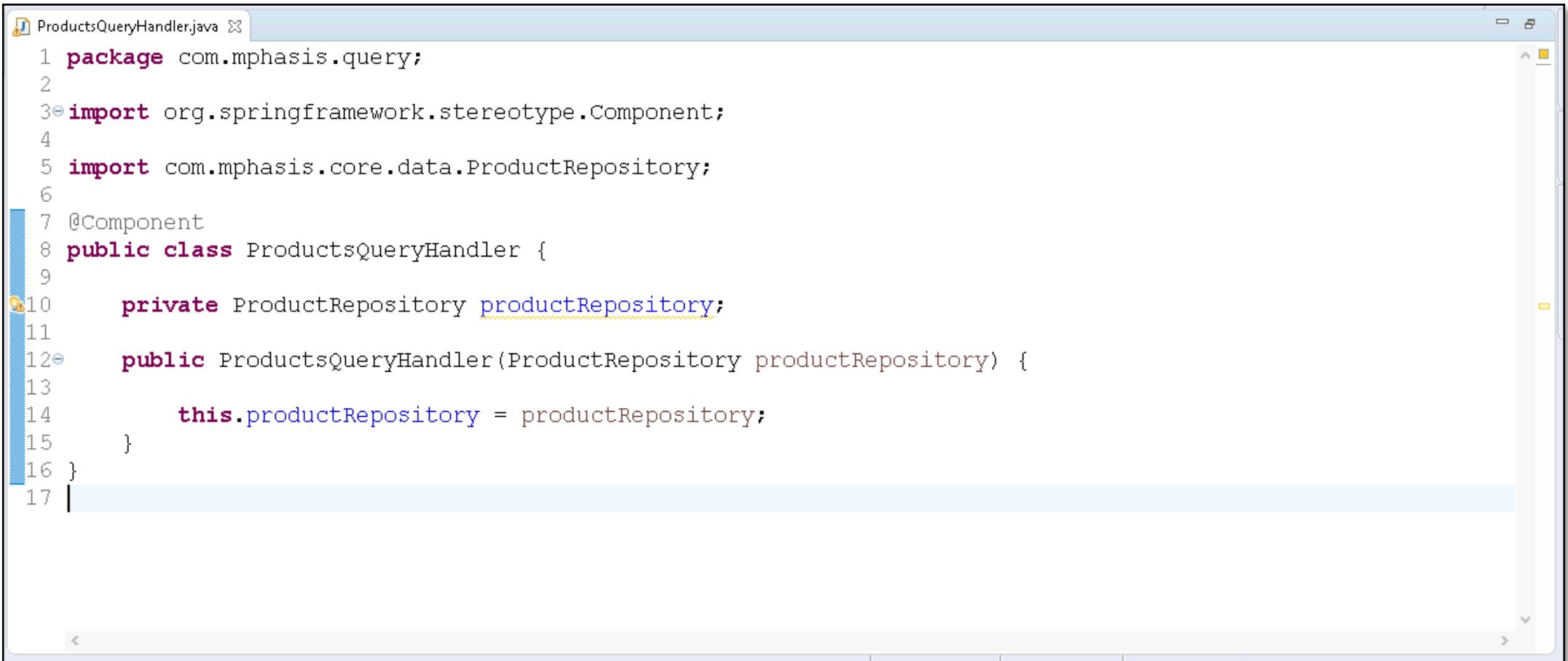


Marker annotation to mark any method on an object as being a QueryHandler.



Create a ProductsQueryHandler class

- Create an ProductQueryHandler class:



The screenshot shows a Java code editor window with the file `ProductsQueryHandler.java` open. The code defines a class named `ProductsQueryHandler` with the following content:

```
1 package com.mphasis.query;
2
3 import org.springframework.stereotype.Component;
4
5 import com.mphasis.core.data.ProductRepository;
6
7 @Component
8 public class ProductsQueryHandler {
9
10     private ProductRepository productRepository;
11
12     public ProductsQueryHandler(ProductRepository productRepository) {
13
14         this.productRepository = productRepository;
15     }
16 }
17
```



Implementing the findProducts() method

- Implementing the findProducts() method:

```
@QueryHandler
public List<ProductRestModel> findProducts(FindProductsQuery query) {

    List<ProductRestModel> productsRest = new ArrayList<>();

    List<ProductEntity> storedProducts = productRepository.findAll();

    for (ProductEntity productEntity : storedProducts) {

        ProductRestModel productRestModel = new ProductRestModel();
        BeanUtils.copyProperties(productEntity, productRestModel);
        productsRest.add(productRestModel);
    }

    return productsRest;
}
```



Trying how it works

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

Send a POST request to Create Product

The screenshot shows the Postman application interface. At the top, there is a navigation bar with 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. Below the navigation bar, a list of requests is shown: 'POST http://localhost:8082/p' and 'GET http://localhost:8082/prc'. A new request card is being edited for 'http://localhost:8082/products-service/products' with a 'POST' method.

The request details panel shows the following configuration:

- Method:** POST
- URL:** http://localhost:8082/products-service/products
- Body:** (8) (highlighted)
- Params:** none
- Authorization:** (disabled)
- Headers:** (8)
- Body Content Type:** JSON (selected)
- Body Data:**

```
1 {  
2   "title": "iPhone 3",  
3   "price": 200,  
4   "quantity": 2  
5 }  
6
```
- Cookies:** (disabled)

The response panel shows the following details:

- Status:** 200 OK
- Time:** 201 ms
- Size:** 152 B
- Save Response:** (button)

The response body is displayed in 'Pretty' format:

```
1 a0d59bd6-d53b-45d2-a0bf-1ec49ad6bbf4
```

At the bottom, there are tabs for 'Console' and other interface elements.

Send a GET request to Query the Products

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. Below the navigation is a list of requests. The first request is a POST to 'http://localhost:8082/p'. The second request, highlighted in green, is a GET to 'http://localhost:8082/products-service/products'. This request has 'GET' selected in the method dropdown and the URL 'http://localhost:8082/products-service/products' in the body field. Below the request details are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', 'Settings', and 'Cookies'. The 'Params' tab is active, showing a table for 'Query Params' with two rows: one for 'Key' and one for 'Value'. Under the 'Body' tab, the response is displayed in 'Pretty' JSON format:

```
1 [  
2   {  
3     "productId": "a0d59bd6-d53b-45d2-a0bf-1ec49ad6bbf4",  
4     "title": "iPhone 3",  
5     "price": 200.00,  
6     "quantity": 2  
7   }  
8 ]
```

The status bar at the bottom indicates 'Status: 200 OK Time: 66 ms Size: 217 B' and a 'Save Response' button. The bottom navigation bar includes 'Body', 'Cookies', 'Headers (3)', 'Test Results', and tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'.

Preview Product record in a Database

The screenshot shows the H2 Console interface. The title bar reads "H2 Console". The address bar indicates a "Not secure" connection to "host.docker.internal:50073/h2-console/login.do?jsessionid=40c59a2c01af425f85d3468af9921240". The toolbar includes buttons for Auto commit (checked), Max rows (set to 1000), Run, Run Selected, Auto complete, Auto select (set to On), and a SQL statement input field containing "SELECT * FROM PRODUCTS". The left sidebar lists database objects: ASSOCIATION_VALUE_ENTRY, PRODUCTS, SAGA_ENTRY, TOKEN_ENTRY, INFORMATION_SCHEMA, Sequences, and Users. A note at the bottom left says "H2 2.1.214 (2022-06-13)". The main content area displays the results of the query:

```
SELECT * FROM PRODUCTS;
```

PRODUCT_ID	PRICE	QUANTITY	TITLE
a0d59bd6-d53b-45d2-a0bf-1ec49ad6bbf4	200.00	2	iPhone 3

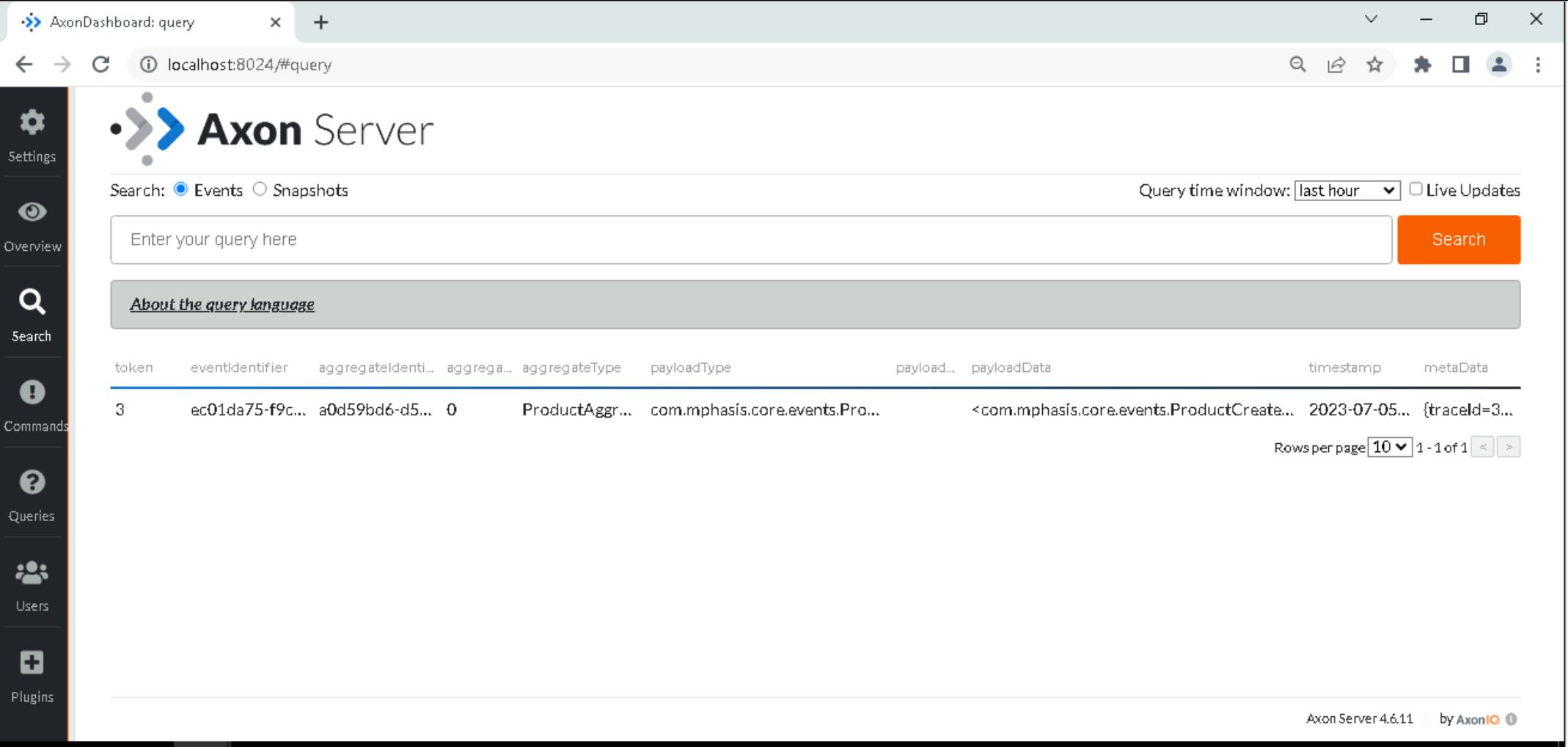
(1 row, 1 ms)

Edit



Previewing Event in the EventStore

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



The screenshot shows the Axon Dashboard interface with the title "Axon Server". The left sidebar has a "Search" tab selected. The main area displays a table of event search results.

Search Options:

- Search type: Events (radio button selected)
- Query time window: last hour
- Live Updates: unchecked

Table Headers:

token	eventIdentifier	aggregateIdent... er	aggregate... er	aggregateType	payloadType	payload... er	payloadData	timestamp	metaData
-------	-----------------	-------------------------	--------------------	---------------	-------------	------------------	-------------	-----------	----------

Table Data:

3	ec01da75-f9c...	a0d59bd6-d5...	0	ProductAggr...	com.mphasis.core.events.Pro...	<com.mphasis.core.events.ProductCreate...	2023-07-05...	{traceId=3...
---	-----------------	----------------	---	----------------	--------------------------------	---	---------------	---------------

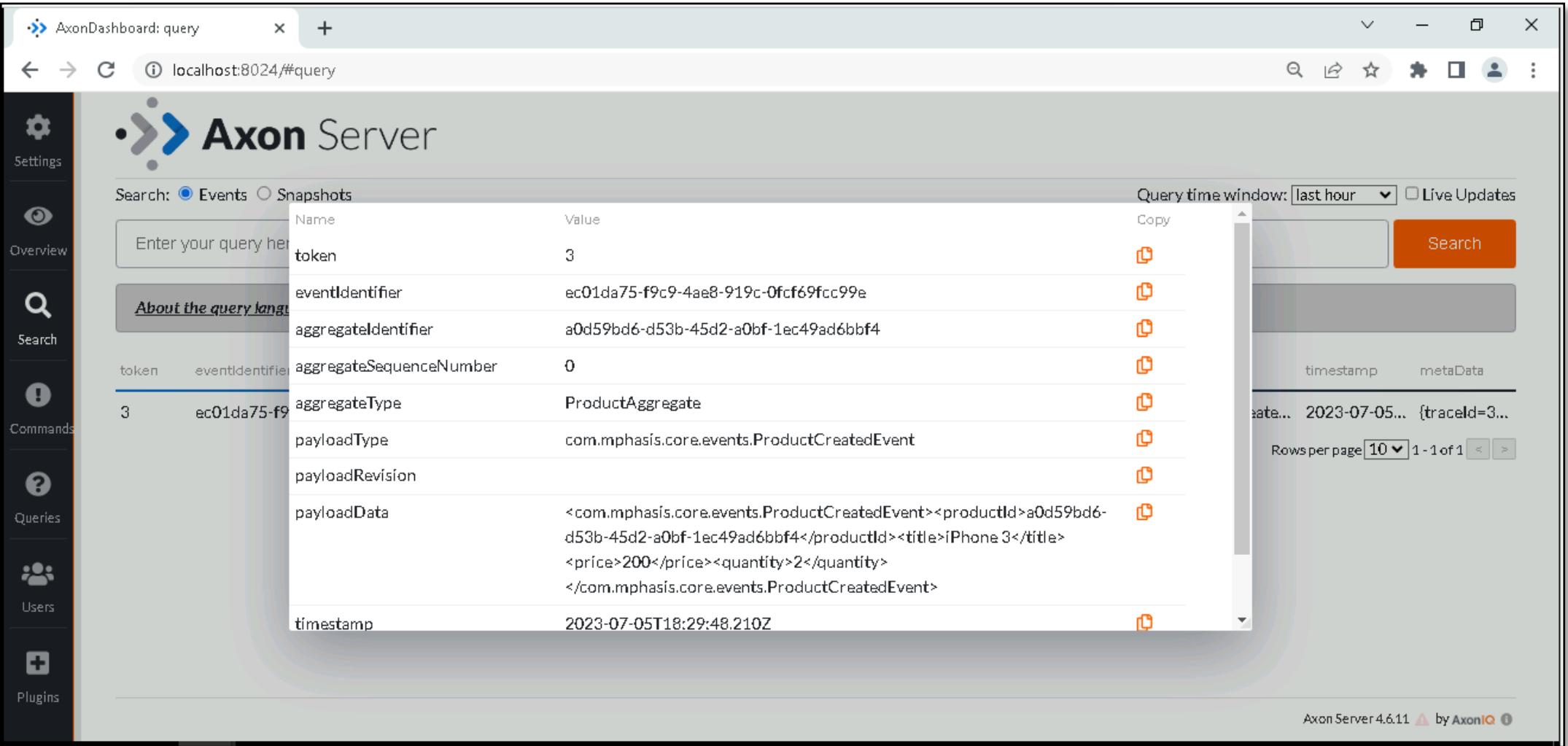
Rows per page: 10 | 1 - 1 of 1 | < >

Axon Server 4.6.11 by AxonIQ



Previewing Event in the EventStore

- View the event details available in Event Store:



The screenshot shows the Axon Server dashboard interface. The left sidebar contains navigation links: Settings, Overview, Search, Commands, Queries, Users, and Plugins. The main area is titled "Axon Server" and displays event details. The search bar at the top says "Search: Events" and has a query input field containing "token". The event details table includes columns for Name and Value. The event identified by token "3" is shown, with its aggregate identifier being "a0d59bd6-d53b-45d2-a0bf-1ec49ad6bbf4". The event type is "ProductAggregate" and the payload type is "com.mphasis.core.events.ProductCreatedEvent". The payload data is XML representing a product creation event with productId "a0d59bd6-d53b-45d2-a0bf-1ec49ad6bbf4", title "iPhone 3", price "200", and quantity "2". The event was timestamped on "2023-07-05T18:29:48.210Z". The right side of the screen shows a preview pane with a timestamp of "2023-07-05..." and a "Rows per page" dropdown set to "10". The bottom right corner of the dashboard footer says "Axon Server 4.6.11 by AxonIQ".

Name	Value
token	3
eventIdentifier	ec01da75-f9c9-4ae8-919c-0fcf69fcc99e
aggregateIdentifier	a0d59bd6-d53b-45d2-a0bf-1ec49ad6bbf4
aggregateSequenceNumber	0
aggregateType	ProductAggregate
payloadType	com.mphasis.core.events.ProductCreatedEvent
payloadRevision	
payloadData	<com.mphasis.core.events.ProductCreatedEvent><productId>a0d59bd6-d53b-45d2-a0bf-1ec49ad6bbf4</productId><title>iPhone 3</title><price>200</price><quantity>2</quantity></com.mphasis.core.events.ProductCreatedEvent>
timestamp	2023-07-05T18:29:48.210Z

- CQRS Persisting Event in the database
- Adding Spring Data JPA & H2 dependencies
- Configure database access in the application.properties file
- Creating the Events Handler/Projection
- Implementing @EventHandler method
- CQRS, Querying Data
- Refactor Command API Rest Controller
- Create a Controller class for Query API
- Get Products Web Service Endpoint
- Implementing @QueryHandler method



Day - 4

- Validating Request Body, Bean Validation
- Bean Validation – with Hibernate Validation
- Bean Validation. Enable Bean Validation
- Bean Validation. Validating Request Body
- Validation in the @CommandHandler method
- Command Validation in the Aggregate
- Introduction to Message Dispatch Interceptor
- Creating a new Command Interceptor class
- Register Message Dispatch Interceptor



Day – 4 Agenda

- Introduction to Set Based Consistency
- Create a Product Lookup Entity
- Create a Product Lookup Repository
- Create a Product Lookup Event Handler
- Persisting information into a ProductLookup table
- Update the MessageDispatchInterceptor class



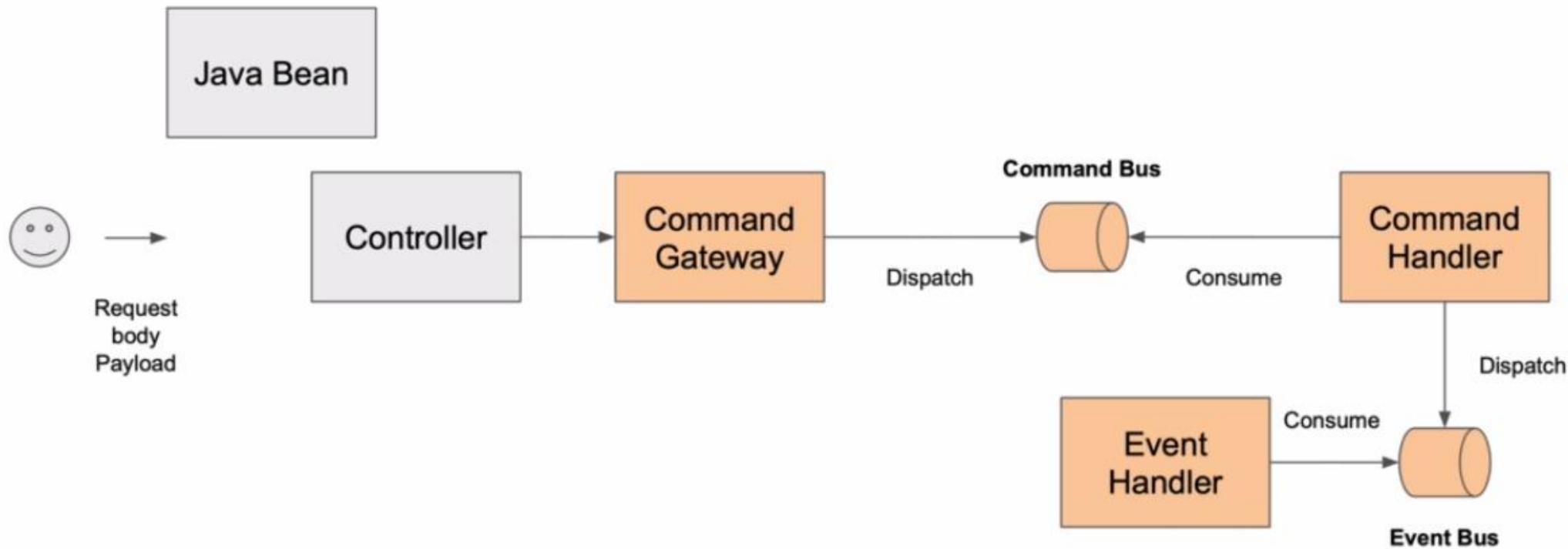
Day - 4

Validating Request Body, Bean Validation



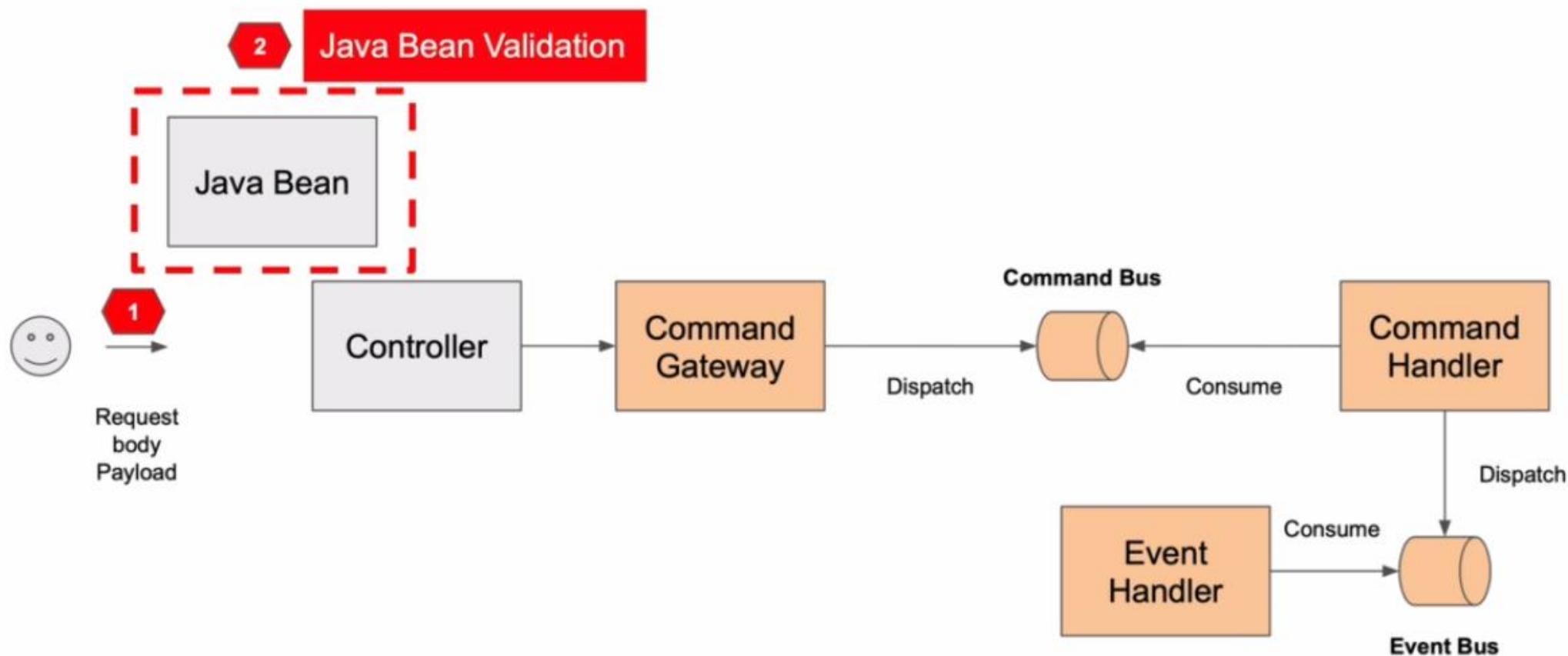
Validating Request Body, Bean Validation

Command API



Validating Request Body, Bean Validation

Command API



```
@Data  
public class CreateProductRestModel {  
  
    @NotNull(message="Product title is a required field")  
    private String title;  
  
    @Min(value=1, message="Price cannot be lower than 1")  
    private BigDecimal price;  
  
    @Min(value=1, message="Quantity cannot be lower than 1")  
    @Max(value=5, message="Quantity cannot be larger than 5")  
    private Integer quantity;  
  
}
```



Bean Validation. Enable Bean Validation

- Add the validation dependency in ProductService/pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```



Bean Validation. Enable Bean Validation

- Add the properties in application.properties file:



```
application.properties X
1
2server.port=0
3eureka.client.service-url.defaultZone=http://localhost:8761/eureka
4spring.application.name=products-service
5eureka.instance.instance-id=${spring.application.name}:${instanceId:${random.value}}
6
7spring.datasource.url=jdbc:h2:mem:mphasisdb
8spring.datasource.driver-class-name=org.h2.Driver
9spring.datasource.username=sa
10spring.datasource.password=password
11spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
12
13#Accessing the H2 Console
14spring.h2.console.enabled=true
15spring.h2.console.path=/h2-console
16spring.h2.console.settings.web-allow-others=true
17
18server.error.include-message=always
19server.error.include-binding-errors=always
20
21|
```



Bean Validation. Validating Request Body

- Add the validation annotation to the fields:

```
1 package com.mphasis.command.rest;
2
3 import java.math.BigDecimal;
4
5
6 @Data
7 public class CreateProductRestModel {
8
9
10    @NotNull(message = "Product title is a required field")
11    private String title;
12
13    @Min(value=1, message = "Price cannot be lower than 1")
14    private BigDecimal price;
15
16    @Min(value=1, message = "Quantity cannot be lower than 1")
17    @Max(value=5, message = "Quantity cannot be larger than 1")
18    private Integer quantity;
19
20
21 }
```



Bean Validation. Validating Request Body

- **@Valid annotation will trigger the validation on the request body:**

```
ProductCommandController.java ✘
25
26  @PostMapping
27  public String createProduct(@Valid @RequestBody CreateProductRestModel createProductRestModel) {
28
29      CreateProductCommand createProductCommand = CreateProductCommand.builder()
30          .price(createProductRestModel.getPrice())
31          .quantity(createProductRestModel.getQuantity())
32          .title(createProductRestModel.getTitle())
33          .productId(UUID.randomUUID().toString())
34          .build();
35
36      String returnValue;
37
38      try {
39          returnValue = commandGateway.sendAndWait(createProductCommand);
40      }catch (Exception ex) {
41          returnValue = ex.getLocalizedMessage();
42      }
43      return returnValue;
44  }
45
```



Trying how it works

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

Send a POST request with title as blank

The screenshot shows the Postman application interface. At the top, there are tabs for 'Home', 'Workspaces', 'Explore', and a search bar. On the right side of the header are 'Sign In' and 'Create Account' buttons. Below the header, there are two requests listed: a 'POST' to 'http://localhost:8082/p...' and a 'GET' to 'http://localhost:8082/pr...'. The main area shows a 'POST' request to 'http://localhost:8082/products-service/products'. The 'Body' tab is selected, showing the following JSON payload:

```
1 {  
2   ... "title": "",  
3   ... "price": 500,  
4   ... "quantity": 2
```

The 'Params', 'Authorization', 'Headers', 'Pre-request Script', 'Tests', and 'Settings' tabs are also visible. Below the body, the response status is shown as 'Status: 400 Bad Request' with a timestamp of 'Time: 67 ms' and a size of 'Size: 714 B'. The 'Pretty' tab is selected in the response panel, displaying the error message:

```
23     },  
24     ],  
25     "defaultMessage": "Product title is a required field",  
26     "objectName": "createProductRestModel",  
27     "field": "title",  
28     "rejectedValue": "",  
29     "bindingFailure": false,  
30     "code": "NotBlank"  
31 }
```

The 'Console' tab is also visible at the bottom left.

Send a POST request with Quantity greater than 20

The screenshot shows the Postman application interface. At the top, there are navigation tabs: Home, Workspaces, Explore, a search bar labeled "Search Postman", and account options: Sign In, Create Account, and a close button.

In the main workspace, there are two requests listed: a POST request to `http://localhost:8082/products-service/products` and a GET request to `http://localhost:8082/products`. The POST request is selected.

The request details are as follows:

- Method:** POST
- URL:** `http://localhost:8082/products-service/products`
- Headers:** (8)
- Body:** (JSON)
- Params:** none
- Authorization:** None
- Tests:** None
- Settings:** None

The JSON body is defined as:

```
1: {
2:   "title": "iPhone 5",
3:   "price": 500,
4:   "quantity": 20
5: }
```

The response status is displayed as:

- Status: 400 Bad Request
- Time: 125 ms
- Size: 712 B
- Save Response

The response body is shown in Pretty, Raw, Preview, and Visualize formats. The response content is:

```
25: [
26:   {
27:     "field": "quantity",
28:     "rejectedValue": 20,
29:     "bindingFailure": false,
30:     "code": "Max"
31:   }
]
```

The "defaultMessage" field is highlighted in blue, indicating it is the error message returned by the server.



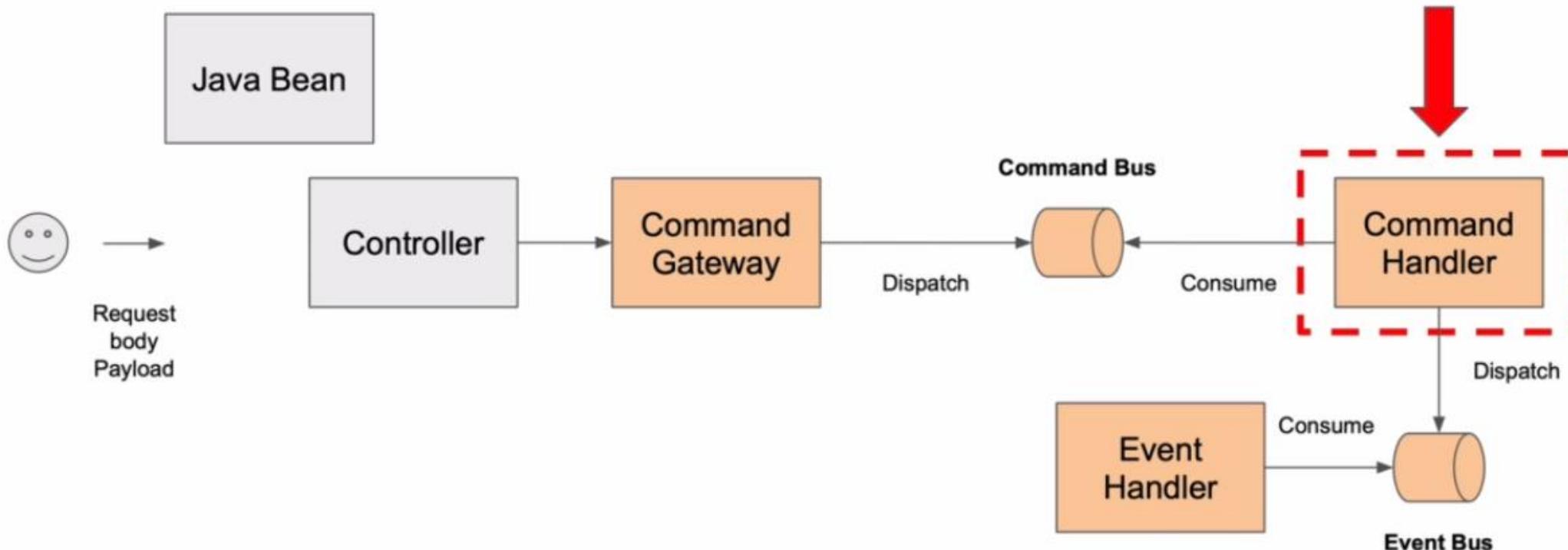
Day - 4

Validation in the @CommandHandler method



Validation in the @CommandHandler method

Command API





Command Validation in the Aggregate

```
@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().isBlank()) {
        throw new IllegalArgumentException("Title cannot be empty");
    }

    ProductCreatedEvent productCreatedEvent = new ProductCreatedEvent();
    BeanUtils.copyProperties(createProductCommand, productCreatedEvent);

    AggregateLifecycle.apply(productCreatedEvent);
}
```



Day - 4

Message Dispatch Interceptor

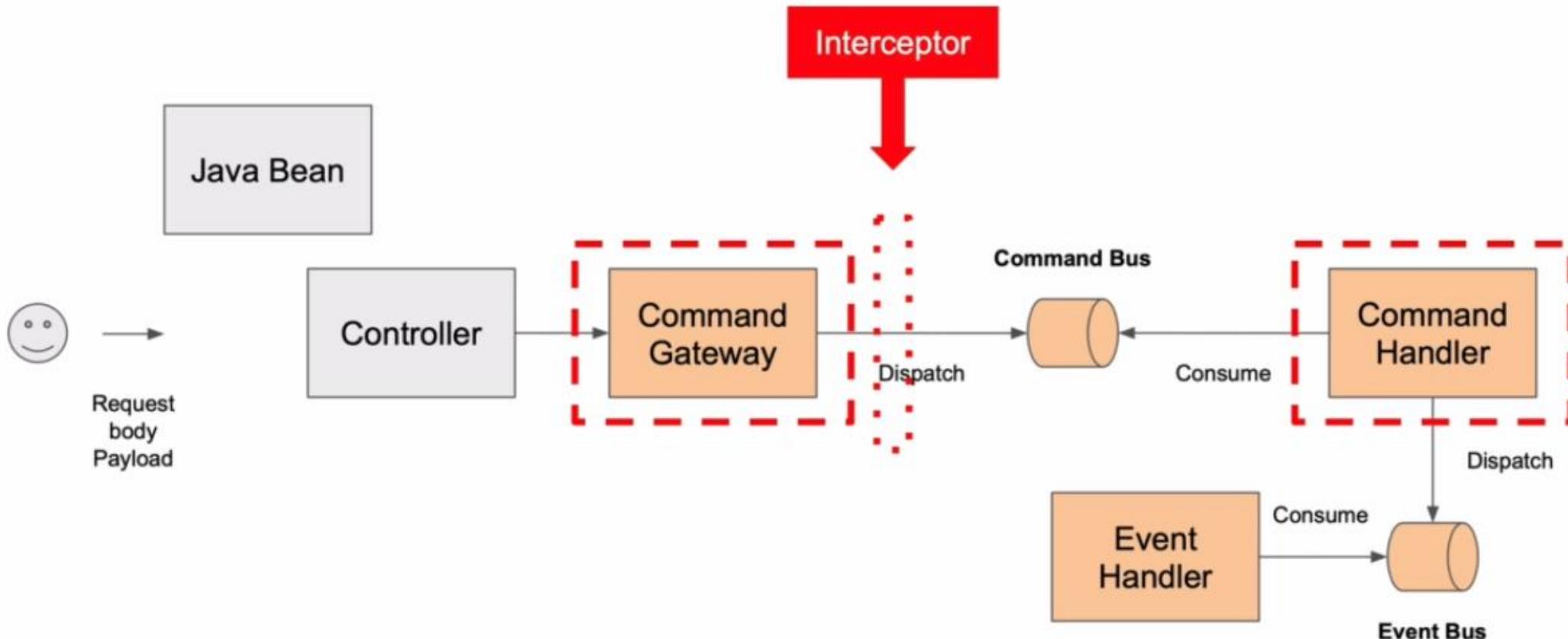


Introduction to Message Dispatch Interceptor

- Message Dispatch Interceptor is invoked when a message is dispatch from Command Gateway to Command Bus.
- We can write a Message Dispatch Interceptor to intercept the command right where it's dispatched on a Command Bus.
- Using Message Dispatch Interceptor, you can do additional logging, you can do command validation, you can alter a command message by adding the META-DATA, you can also block the command by throwing an Exception.

Introduction to Message Dispatch Interceptor

Command API





Creating a new Command Interceptor class

- Create a new CreateProductCommandInterceptor class:

```
15 @Component
16 public class CreateProductCommandInterceptor implements MessageDispatchInterceptor<CommandMessage<?>>{
17
18     private static final Logger LOGGER = LoggerFactory.getLogger(CreateProductCommandInterceptor.class);
19
20     @Override
21     public BiFunction<Integer, CommandMessage<?>, CommandMessage<?>> handle(
22         List<? extends CommandMessage<?>> messages) {
23
24         return (index, command) -> {
25
26             LOGGER.info("Intercepted command: " + command.getPayloadType());
27
28             if(CreateProductCommand.class.equals(command.getPayloadType())) {
29
30                 CreateProductCommand createProductCommand = (CreateProductCommand) command.getPayload();
31
32                 if (createProductCommand.getPrice().compareTo(BigDecimal.ZERO) <= 0) {
33                     throw new IllegalArgumentException("Price cannot be less or equal than zero");
34                 }
35
36                 if (createProductCommand.getTitle() == null || createProductCommand.getTitle().isEmpty()) {
37                     throw new IllegalArgumentException("Title cannot be empty");
38                 }
39             }
40             return command;
41         };
42     }
}
```



Register Message Dispatch Interceptor

- Register the CreateProductCommandInterceptor class:



```
ProductServiceApplication.java
1 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
2 import org.springframework.context.ApplicationContext;
3
4 import com.mphasis.command.interceptor.CreateProductCommandInterceptor;
5
6 @EnableDiscoveryClient
7 @SpringBootApplication
8 public class ProductServiceApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(ProductServiceApplication.class, args);
12     }
13
14     @Autowired
15     public void registerCreateProductCommandInterceptor(
16         ApplicationContext context, CommandBus commandBus) {
17
18         commandBus.registerDispatchInterceptor(context.getBean(CreateProductCommandInterceptor.class));
19     }
20 }
```



Trying how it works

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

Send a POST request with title as blank

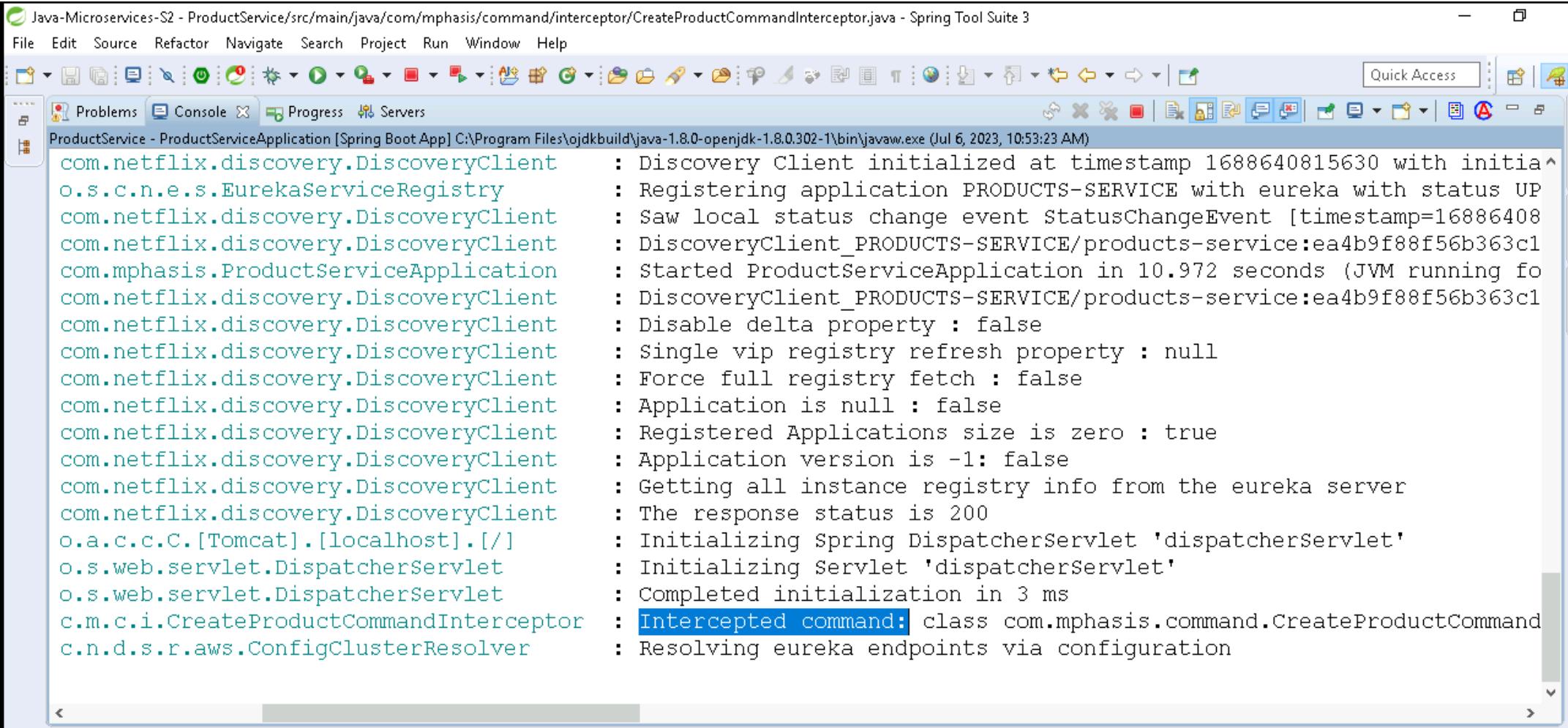
The screenshot shows the Postman application interface. At the top, there is a navigation bar with 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. Below the navigation bar, a header bar shows a 'POST' request to 'http://localhost:8082/p...' and a 'GET' request to 'http://localhost:8082/pr...'. The main workspace displays a POST request to 'http://localhost:8082/products-service/products'. The 'Body' tab is selected, showing a JSON payload with an empty 'title' field:

```
1 {  
2   ... "title": "",  
3   ... "price": 500,  
4   ... "quantity": 5  
5 }  
6
```

The 'Pretty' tab in the Body panel is selected, showing the JSON with line numbers and indentation. Below the body panel, the status bar indicates 'Status: 200 OK Time: 97 ms Size: 137 B'. The bottom left corner shows a 'Console' tab.



Check the ProductServiceApplication Console



The screenshot shows the Spring Tool Suite 3 interface with the 'Console' tab selected. The output window displays log messages from the ProductServiceApplication. A specific message is highlighted in blue:

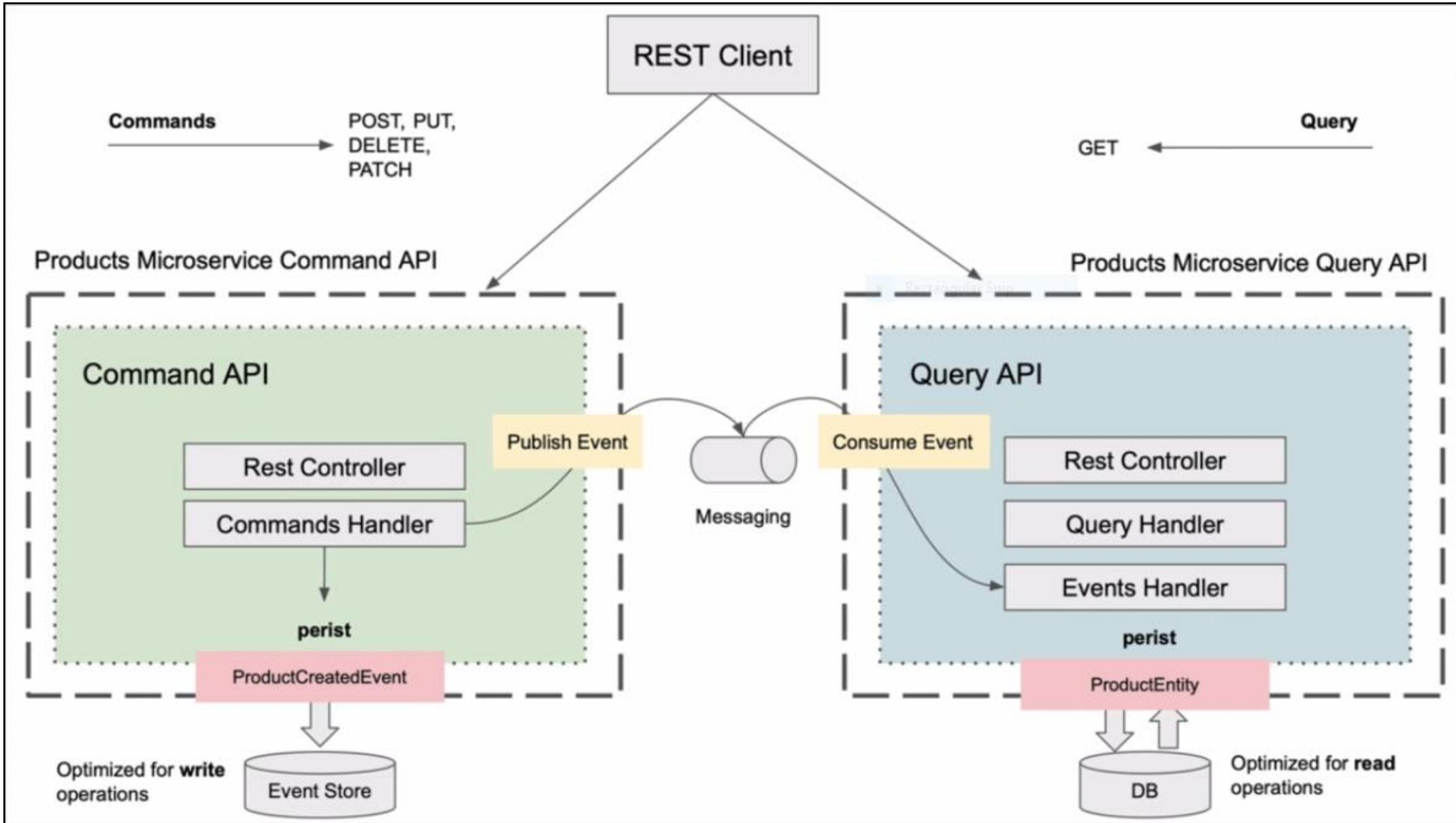
```
com.netflix.discovery.DiscoveryClient : Discovery Client initialized at timestamp 1688640815630 with initia^
o.s.c.n.e.s.EurekaServiceRegistry : Registering application PRODUCTS-SERVICE with eureka with status UP
com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent [timestamp=16886408
com.netflix.discovery.DiscoveryClient : DiscoveryClient_PRODUCTS-SERVICE/products-service:ea4b9f88f56b363c1
com.mphasis.ProductServiceApplication : Started ProductServiceApplication in 10.972 seconds (JVM running fo
com.netflix.discovery.DiscoveryClient : DiscoveryClient_PRODUCTS-SERVICE/products-service:ea4b9f88f56b363c1
com.netflix.discovery.DiscoveryClient : Disable delta property : false
com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : null
com.netflix.discovery.DiscoveryClient : Force full registry fetch : false
com.netflix.discovery.DiscoveryClient : Application is null : false
com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : true
com.netflix.discovery.DiscoveryClient : Application version is -1: false
com.netflix.discovery.DiscoveryClient : Getting all instance registry info from the eureka server
com.netflix.discovery.DiscoveryClient : The response status is 200
o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms
c.m.c.i.CreateProductCommandInterceptor : Intercepted command: class com.mphasis.command.CreateProductCommand
c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
```



Day - 4

Set Based Consistency

Introduction to Set Based Consistency





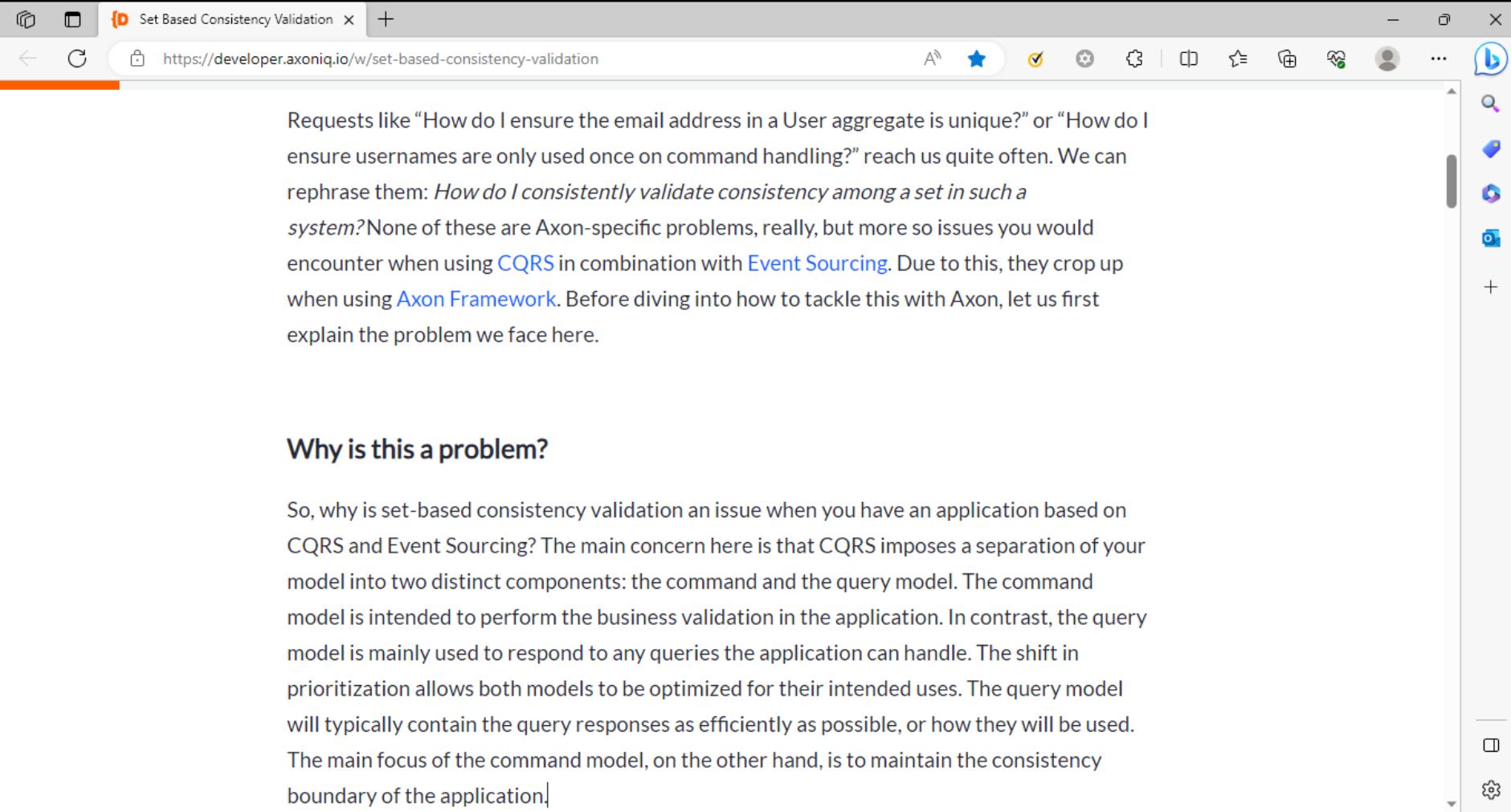
Introduction to Set Based Consistency

- How to check if record already exists in a database table?
 - How to check if Product already exists?
 - How to check if User already exists?
-
- Communication between Command API and Query API is via **Messaging Architecture**.
 - How can Command API quickly check the record already exists for the Persistent Event in the Event Store.



Introduction to Set Based Consistency

- Go to <https://developer.axoniq.io/w/set-based-consistency-validation>



The screenshot shows a Microsoft Edge browser window with the title "Set Based Consistency Validation". The URL in the address bar is "https://developer.axoniq.io/w/set-based-consistency-validation". The main content area contains the following text:

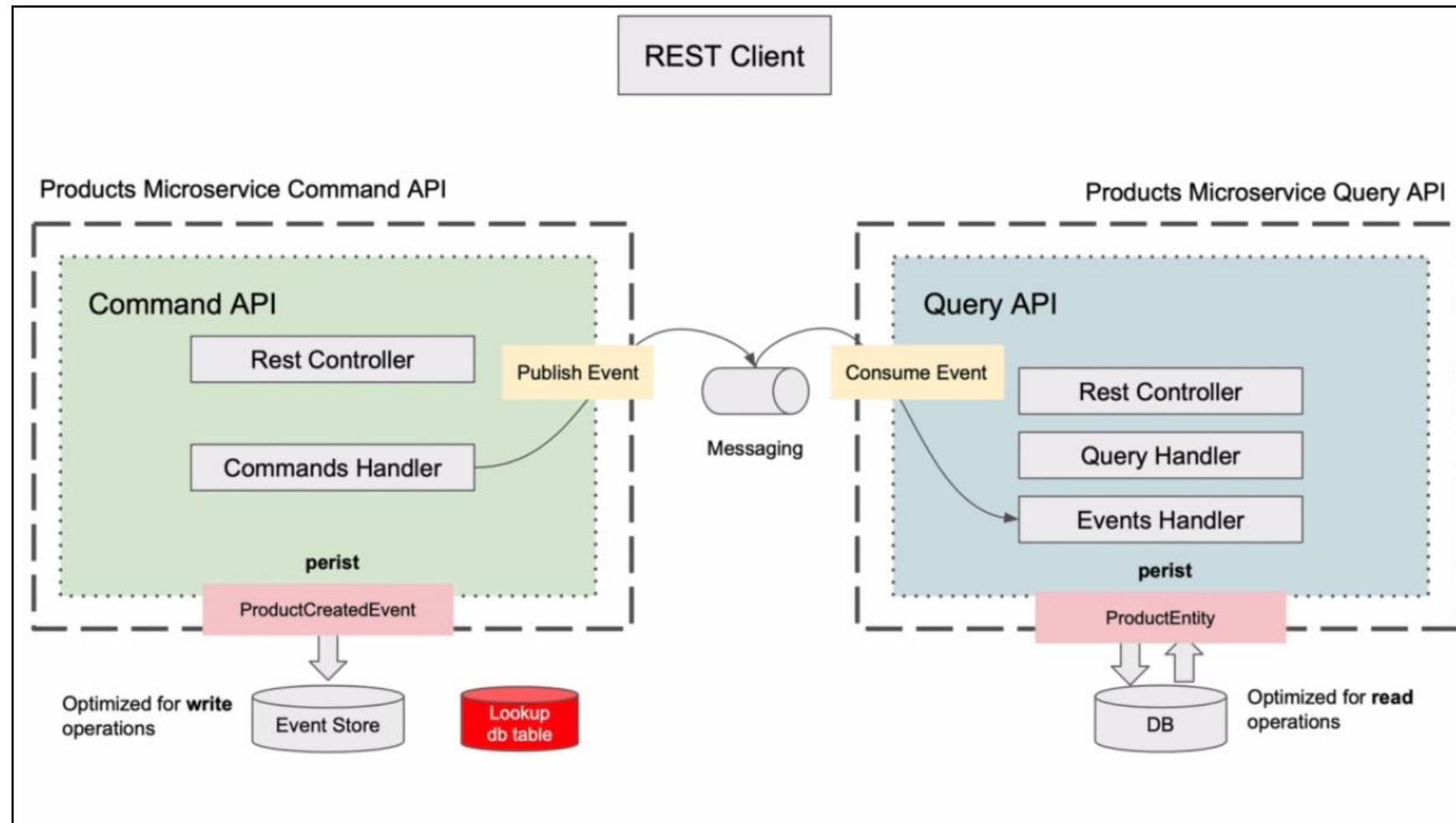
Requests like "How do I ensure the email address in a User aggregate is unique?" or "How do I ensure usernames are only used once on command handling?" reach us quite often. We can rephrase them: *How do I consistently validate consistency among a set in such a system?* None of these are Axon-specific problems, really, but more so issues you would encounter when using CQRS in combination with Event Sourcing. Due to this, they crop up when using Axon Framework. Before diving into how to tackle this with Axon, let us first explain the problem we face here.

Why is this a problem?

So, why is set-based consistency validation an issue when you have an application based on CQRS and Event Sourcing? The main concern here is that CQRS imposes a separation of your model into two distinct components: the command and the query model. The command model is intended to perform the business validation in the application. In contrast, the query model is mainly used to respond to any queries the application can handle. The shift in prioritization allows both models to be optimized for their intended uses. The query model will typically contain the query responses as efficiently as possible, or how they will be used. The main focus of the command model, on the other hand, is to maintain the consistency boundary of the application.

Set Based Consistency

- Because the command model can contain any form of data model. We will introduce a small lookup table which will contain product IDs and product titles.





Lookup Table

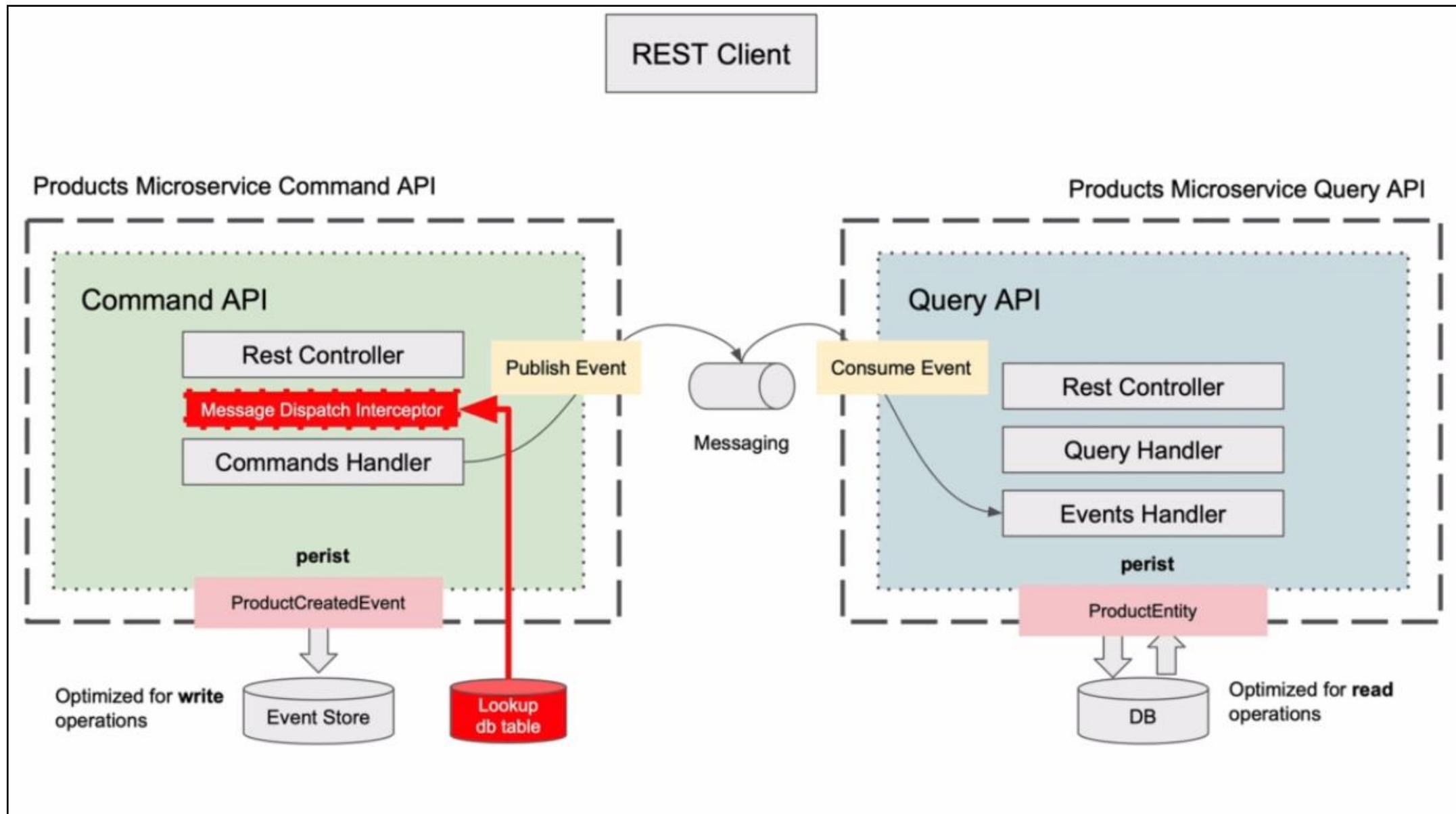
- It should not contain the exact same product details as the read database has.
- This lookup table should only store product fields that are needed to look up the record and see if it exists. Like for example, productID or the product title.
- No need to store their product price, product quantity or other product details.
- If the client application issues a command to update product title, for example, and the product title is the field by which you look up a record in this look up database table then you will need to update this field in both look up table and your query database by which you query the record in the lookup table must be consistent with the main read product database.



Set Based Consistency

- How do we query this lookup table before the command handler processes the command?
- And the answer is, we use Message Dispatch Interceptor which we have already implemented.

Set Based Consistency

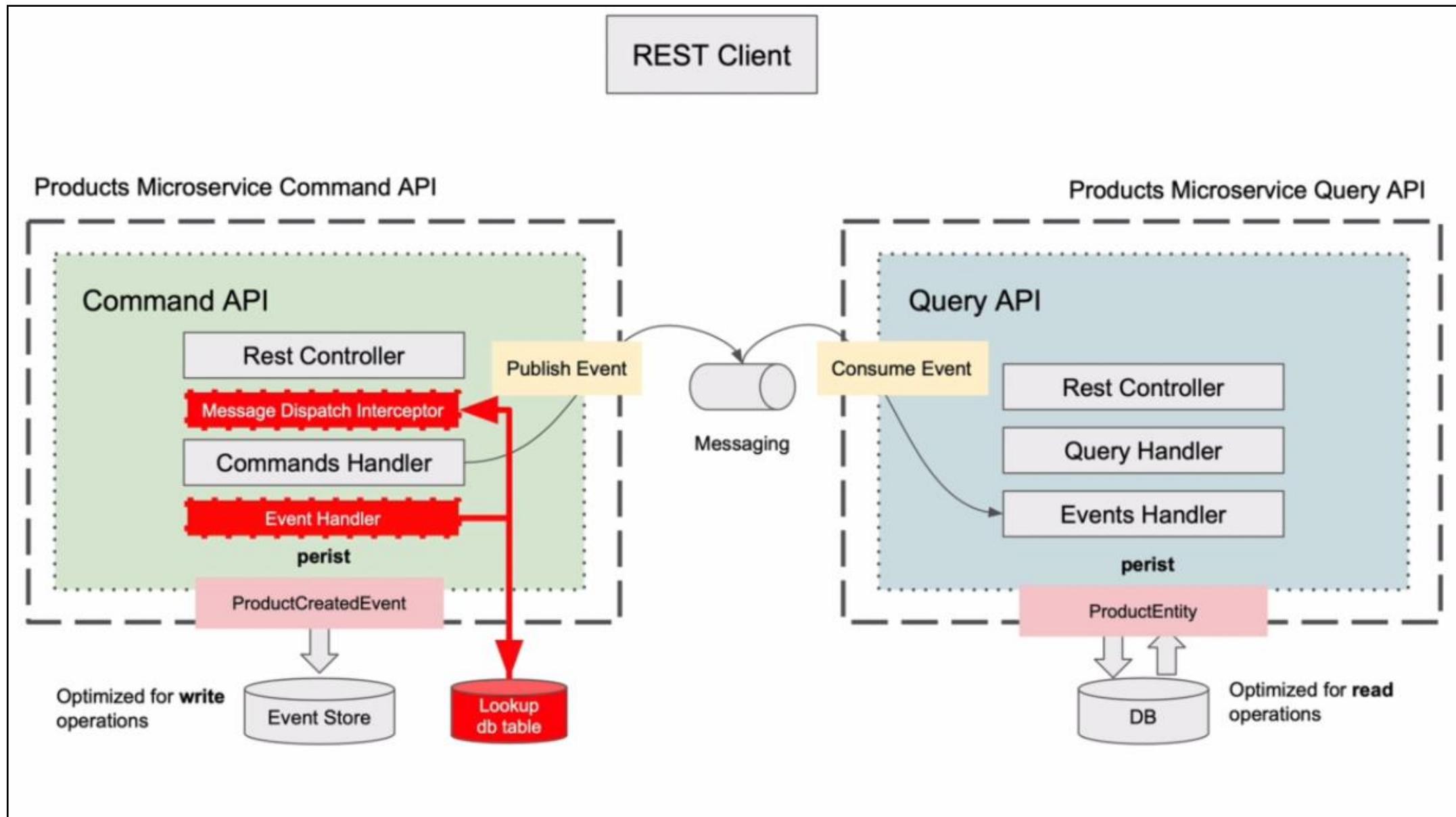




Set Based Consistency

- How do we will write into this lookup table?
- We will implement an additional **event handler** that will persist product ID and the product title into this database table and when the command handler method publishes the product created event.

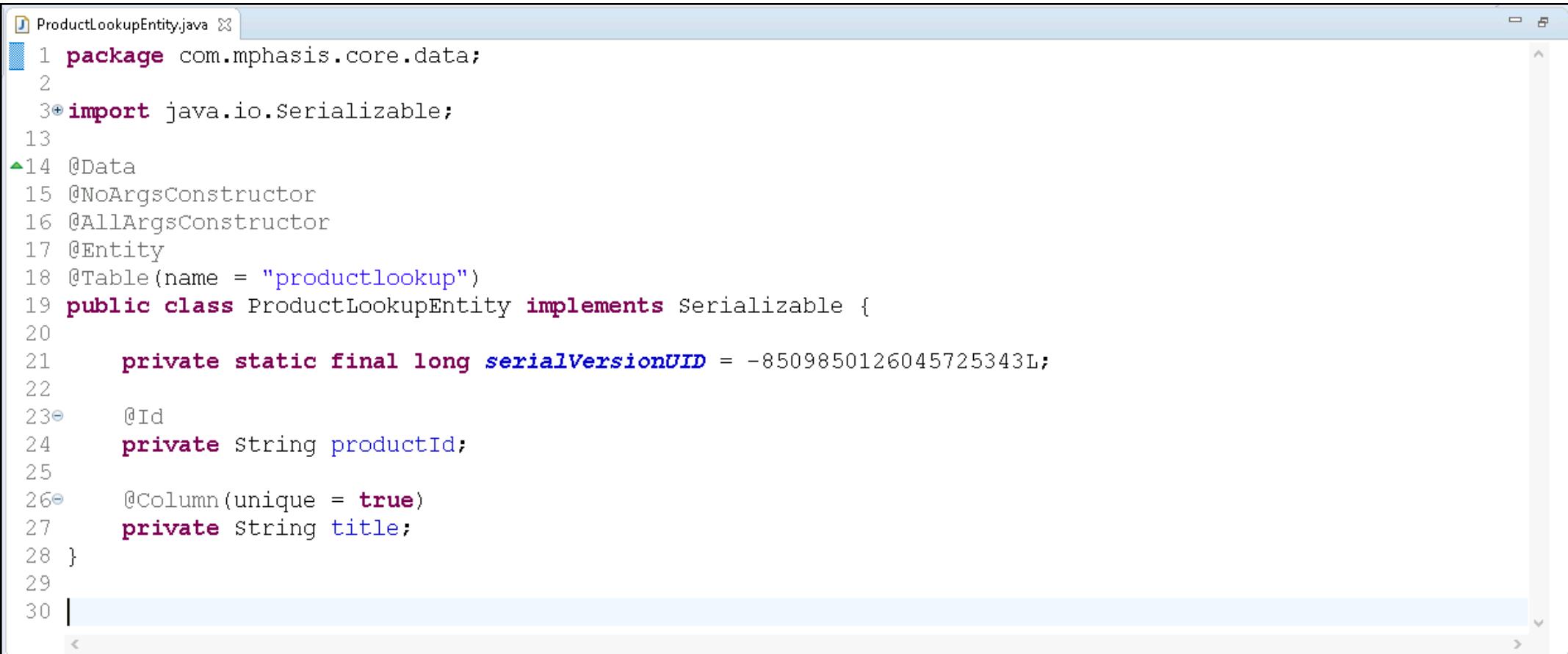
Set Based Consistency





Create a ProductLookupEntity class

- Create a ProductLookupEntity class:



The screenshot shows a Java code editor window with the file `ProductLookupEntity.java` open. The code defines a class `ProductLookupEntity` that implements the `Serializable` interface. The class uses Lombok annotations: `@Data`, `@NoArgsConstructor`, `@AllArgsConstructor`, `@Entity`, and `@Table(name = "productlookup")`. It contains two private fields: `productId` and `title`, both annotated with `@Id` and `@Column(unique = true)` respectively. A static final long variable `serialVersionUID` is also defined.

```
1 package com.mphasis.core.data;
2
3 import java.io.Serializable;
4
5 @Data
6 @NoArgsConstructor
7 @AllArgsConstructor
8 @Entity
9 @Table(name = "productlookup")
10 public class ProductLookupEntity implements Serializable {
11
12     private static final long serialVersionUID = -8509850126045725343L;
13
14     @Id
15     private String productId;
16
17     @Column(unique = true)
18     private String title;
19 }
```



Create a ProductLookupRepository class

- Create a ProductLookupRepository class:

The screenshot shows a Java code editor window with the title "ProductLookupRepository.java". The code defines a public interface named "ProductLookupRepository" that extends "JpaRepository<ProductLookupEntity, String>". It contains a single method, "findByProductIdOrTitle", which takes two String parameters: "productId" and "title".

```
1 package com.mphasis.core.data;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface ProductLookupRepository extends JpaRepository<ProductLookupEntity, String>{
6
7     ProductLookupEntity findByProductIdOrTitle(String productId, String title);
8 }
9 |
```



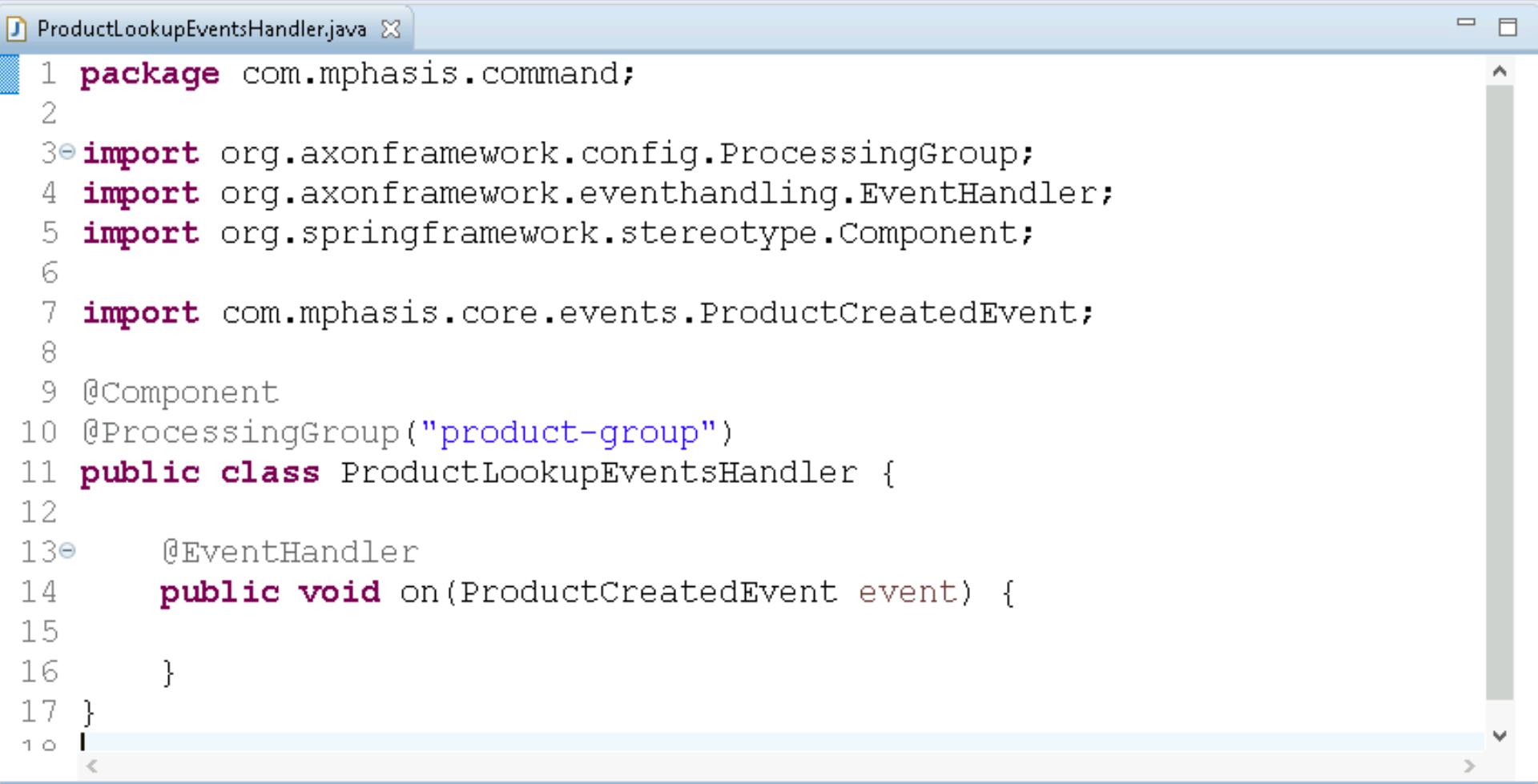
Create a ProductLookupEventsHandler class

- Should be annotated with **@ProcessingGroup("product-group")**
- This annotation helps in logically group the events handler together.
- Will group both the events handler class – ProductLookupEventsHandler and ProductEventsHandler.
- For this ProcessingGroup, the Axon will also create separate its own TrackingEventProcessor.
- The **TrackingEventProcessor** will use a special tracking token which it will use to avoid multiple processing of the same event in different threads and nodes.
- So, by using this processing group annotation and by grouping these two event handlers in the same logical group, we will also make sure that both events are handled only once and that they're handled in the same thread.
- And this also gives us possibility to roll back the whole transaction if event processing is not successful.



Create a ProductLookupEventsHandler class

- Create a ProductLookupEventsHandler class:

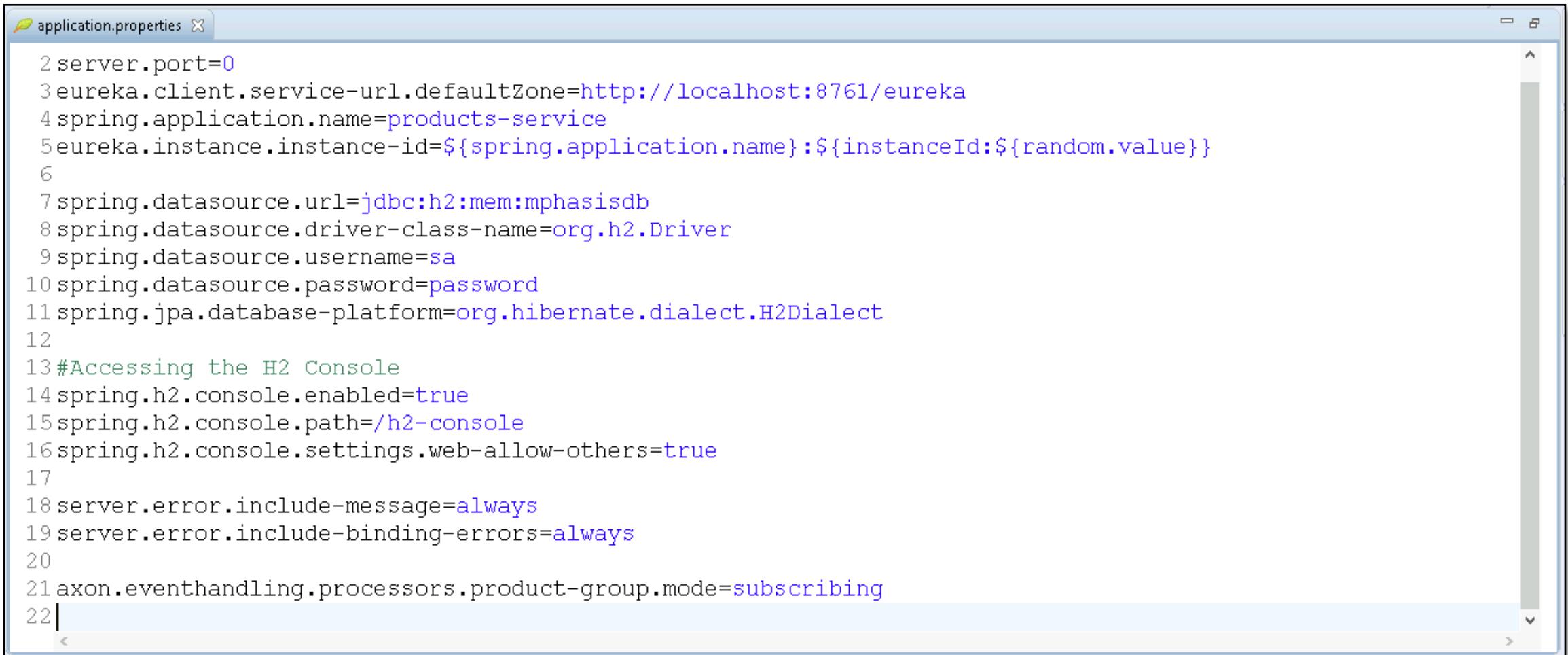


The screenshot shows a Java code editor window with the file `ProductLookupEventsHandler.java` open. The code defines a class named `ProductLookupEventsHandler` that implements the `EventHandler` interface for the `ProductCreatedEvent`. The class is annotated with `@Component` and `@ProcessingGroup("product-group")`.

```
1 package com.mphasis.command;
2
3 import org.axonframework.config.ProcessingGroup;
4 import org.axonframework.eventhandling.EventHandler;
5 import org.springframework.stereotype.Component;
6
7 import com.mphasis.core.events.ProductCreatedEvent;
8
9 @Component
10 @ProcessingGroup("product-group")
11 public class ProductLookupEventsHandler {
12
13     @EventHandler
14     public void on(ProductCreatedEvent event) {
15
16     }
17 }
```



Add the property to application.properties file



```
application.properties
2 server.port=0
3 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
4 spring.application.name=products-service
5 eureka.instance.instance-id=${spring.application.name}:${instanceId:${random.value}}
6
7 spring.datasource.url=jdbc:h2:mem:mphasisdb
8 spring.datasource.driver-class-name=org.h2.Driver
9 spring.datasource.username=sa
10 spring.datasource.password=password
11 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
12
13 #Accessing the H2 Console
14 spring.h2.console.enabled=true
15 spring.h2.console.path=/h2-console
16 spring.h2.console.settings.web-allow-others=true
17
18 server.error.include-message=always
19 server.error.include-binding-errors=always
20
21 axon.eventhandling.processors.product-group.mode=subscribing
22 |
```



Persisting information into a ProductLookup table

- Persisting information into a ProductLookup table:

The screenshot shows a Java code editor window with the file `ProductLookupEventsHandler.java` open. The code defines a component named `ProductLookupEventsHandler` that handles `ProductCreatedEvent` by persisting product information into a `ProductLookupRepository`. The code uses Lombok annotations for the constructor and the event handler method.

```
10
11 @Component
12 @ProcessingGroup("product-group")
13 public class ProductLookupEventsHandler {
14
15     private final ProductLookupRepository productLookupRepository;
16
17     public ProductLookupEventsHandler(ProductLookupRepository productLookupRepository) {
18         this.productLookupRepository = productLookupRepository;
19     }
20
21     @EventHandler
22     public void on(ProductCreatedEvent event) {
23
24         ProductLookupEntity productLookupEntity = new ProductLookupEntity(
25             event.getProductId(), event.getTitle());
26
27         productLookupRepository.save(productLookupEntity);
28     }
29 }
30
```



Update the MessageDispatchInterceptor class

- Let's remove the if conditions of Price & Title from the CreateProductCommandInterceptor class:

```
▲28  public BiFunction<Integer, CommandMessage<?>, CommandMessage<?>> handle(
29      List<? extends CommandMessage<?>> messages) {
30
31      return (index, command) -> {
32
33          LOGGER.info("Intercepted command: " + command.getPayloadType());
34
35          if(CreateProductCommand.class.equals(command.getPayloadType())) {
36
37              CreateProductCommand createProductCommand = (CreateProductCommand) command.getPayload();
38
39              ProductLookupEntity productLookupEntity = productLookupRepository.findByIdOrTitle(
40                  createProductCommand.getProductId(), createProductCommand.getTitle());
41
42              if(productLookupEntity != null) {
43                  throw new IllegalStateException(
44                      String.format("Product with productId %s or title %s already exist",
45                      createProductCommand.getProductId(),
46                      createProductCommand.getTitle())
47                  );
48              }
49          }
50      }
51  };
```



Trying how it works

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

Send a POST request to Create Product

The screenshot shows the Postman application interface. At the top, there is a navigation bar with 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. Below the navigation bar, a header bar displays 'POST http://localhost:8082/p...' and 'GET http://localhost:8082/prc...'. The main workspace shows an 'HTTP' request to 'http://localhost:8082/products-service/products'. The method is set to 'POST' and the URL is 'http://localhost:8082/products-service/products'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "title": "iPhone 8",  
3   "price": 500,  
4   "quantity": 5  
5 }  
6
```

The 'Params', 'Authorization', 'Headers', 'Pre-request Script', 'Tests', and 'Settings' tabs are also visible. Below the body, the response status is shown as 'Status: 200 OK Time: 446 ms Size: 152 B'. The response body contains the ID: '6f0fe190-ef02-48ef-93a7-b0bd17524c46'. The bottom of the interface includes a 'Console' tab.

Send a GET request to Query the Products

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. Below the navigation is a list of requests: 'POST http://localhost:8082/p' (disabled) and 'GET http://localhost:8082/pro'. A new request button '+' and an ellipsis '...' are also present.

The main area displays a request configuration for a 'GET' method to 'http://localhost:8082/products-service/products'. The 'Params' tab is selected, showing two query parameters: 'Key' (Value: 'Value') and another 'Key' (Value: 'Value'). The 'Headers' tab shows six headers: 'Content-Type: application/json', 'Accept: application/json', 'User-Agent: PostmanRuntime/7.31.0', 'Host: localhost:8082', 'Connection: keep-alive', and 'Cache-Control: no-cache'. The 'Body' tab is selected, showing a JSON response:

```
1 [  
2   {  
3     "productId": "6f0fe190-ef02-48ef-93a7-b0bd17524c46",  
4     "title": "iPhone 8",  
5     "price": 500.00,  
6     "quantity": 5  
7   }  
8 ]
```

The status bar at the bottom indicates the response was successful: 'Status: 200 OK' with a duration of 'Time: 57 ms' and a size of 'Size: 217 B'. There's also a 'Save Response' button. The bottom left corner shows a 'Console' tab.

Product Lookup Table

The screenshot shows the H2 Console interface. The title bar indicates the connection is "Not secure" to "host.docker.internal:49967/h2-console/login.do?jsessionid=6fe629166adaee826763662aa3da238e". The left sidebar lists database objects: ASSOCIATION_VALUE_ENTRY, PRODUCTLOOKUP, PRODUCTS, SAGA_ENTRY, TOKEN_ENTRY, INFORMATION_SCHEMA, Sequences, and Users. The main area contains the SQL statement "SELECT * FROM PRODUCTLOOKUP" and its execution results.

SQL statement:

```
SELECT * FROM PRODUCTLOOKUP;
```

Execution results:

PRODUCT_ID	TITLE
6f0fe190-ef02-48ef-93a7-b0bd17524c46	iPhone 8

(1 row, 6 ms)

Buttons: Run, Run Selected, Auto complete, Clear, SQL statement.

Send a POST request to Create Product with same JSON

The screenshot shows the Postman application interface. At the top, there is a navigation bar with 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. Below the navigation bar, a header bar displays 'POST http://localhost:8082/p' and 'GET http://localhost:8082/pr'. The main workspace shows a POST request to 'http://localhost:8082/products-service/products'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   ... "title": "iPhone 8",  
3   ... "price": 500,  
4   ... "quantity": 5  
5 }  
6
```

The 'Body' tab also includes tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'Text'. The response section shows a status of '200 OK', time '18 ms', size '207 B', and a message: 'Product with productId 76e3599e-d922-4e01-8190-4272b93ba4cb or title iPhone 8 already exist'.



Recap of Day – 4

- Validating Request Body, Bean Validation
- Bean Validation – with Hibernate Validation
- Bean Validation. Enable Bean Validation
- Bean Validation. Validating Request Body
- Validation in the @CommandHandler method
- Command Validation in the Aggregate
- Introduction to Message Dispatch Interceptor
- Creating a new Command Interceptor class
- Register Message Dispatch Interceptor



Recap of Day – 4

- Introduction to Set Based Consistency
- Create a Product Lookup Entity
- Create a Product Lookup Repository
- Create a Product Lookup Event Handler
- Persisting information into a ProductLookup table
- Update the MessageDispatchInterceptor class



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 ($C=X^2C^2$) 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

This document is the property of, and is proprietary to Mphasis, and identified as "Confidential". Those parties to whom it is distributed shall exercise the same degree of custody and care afforded their own such information. It is not to be disclosed, in whole or in part to any third parties, without the express written authorization of Mphasis. It is not to be duplicated or used, in whole or in part, for any purpose other than the evaluation of, and response to, Mphasis' proposal or bid, or the performance and execution of a contract awarded to Mphasis. This document will be returned to Mphasis upon request.



Any Questions?

Manpreet.Bindra@mphasis.com

FOLLOW US IN THE LINK BELOW

@Mphasis

