

Java Microservices Hands-on Mastery - S1

Manpreet Singh Bindra
Senior Manager



Do's and Don't

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



Welcome

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



Overall Agenda

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



Day - 1



Day – 1 Agenda

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



Day - 1

Microservice vs Monolithic Application



What are Microservice?

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

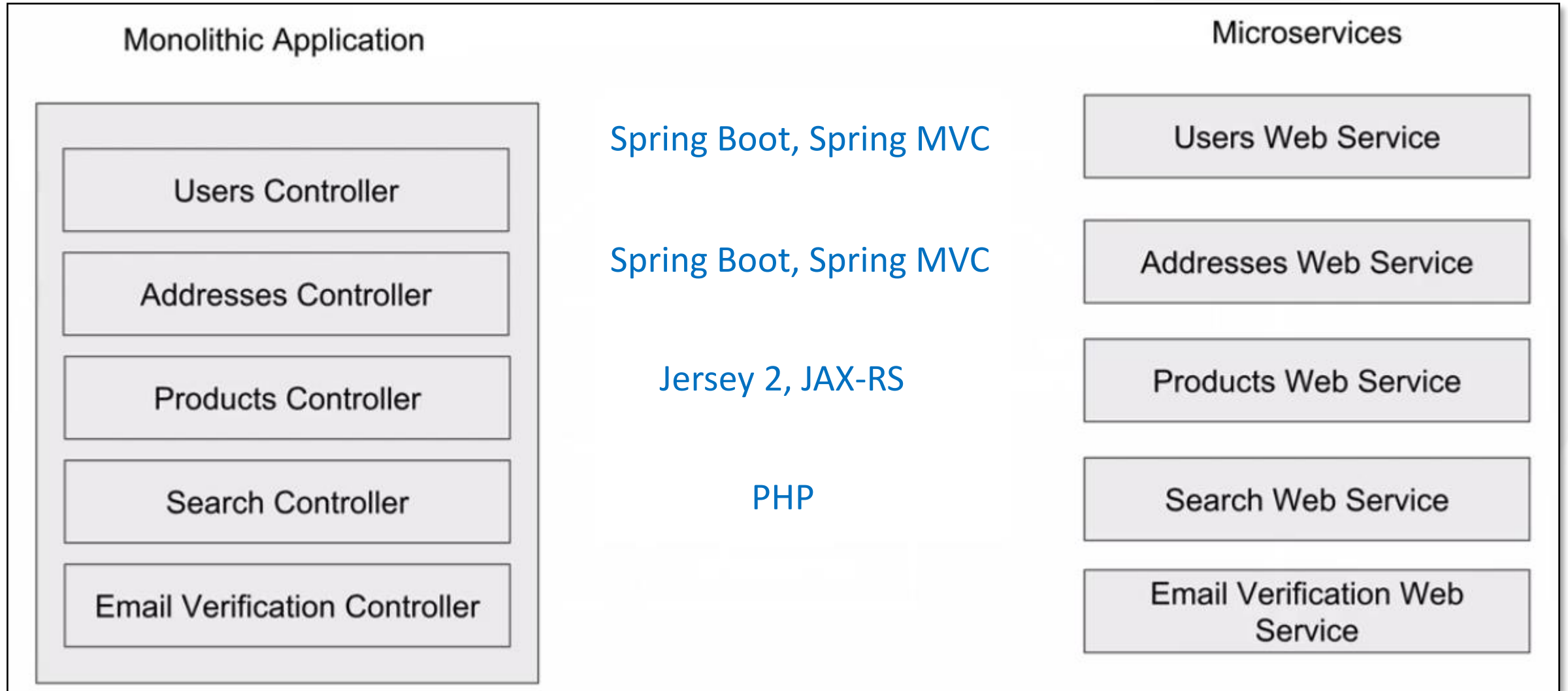


What are Microservice?

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

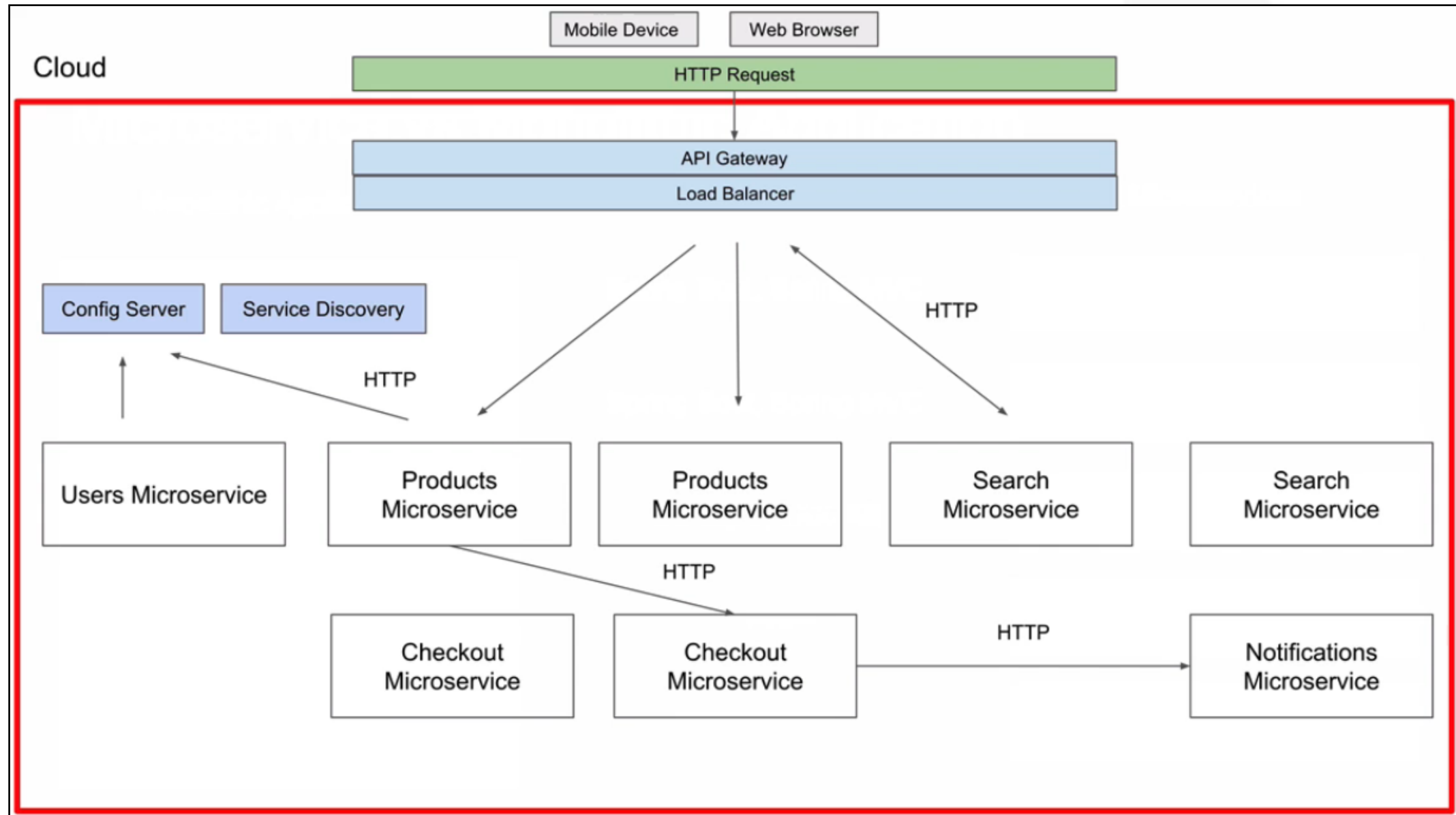


Microservice vs Monolithic Application





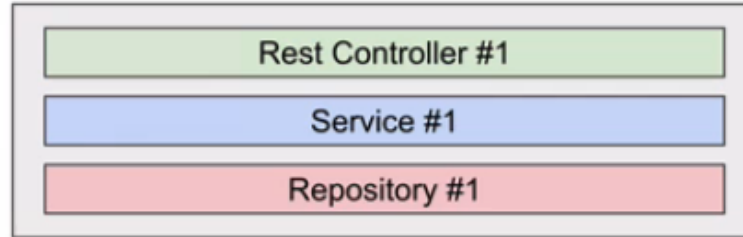
Application Diagram



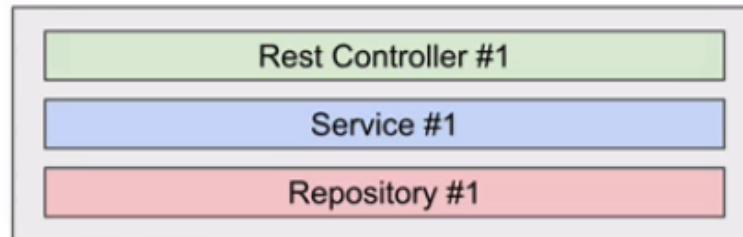


Rest Controller - Services

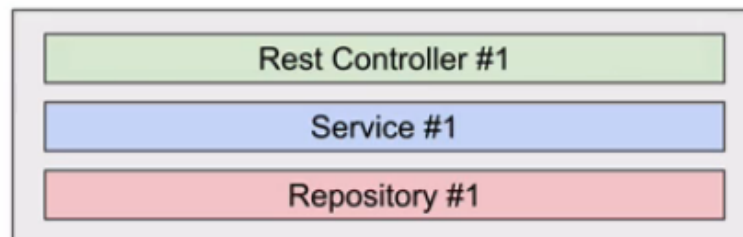
Users



Addresses



Search



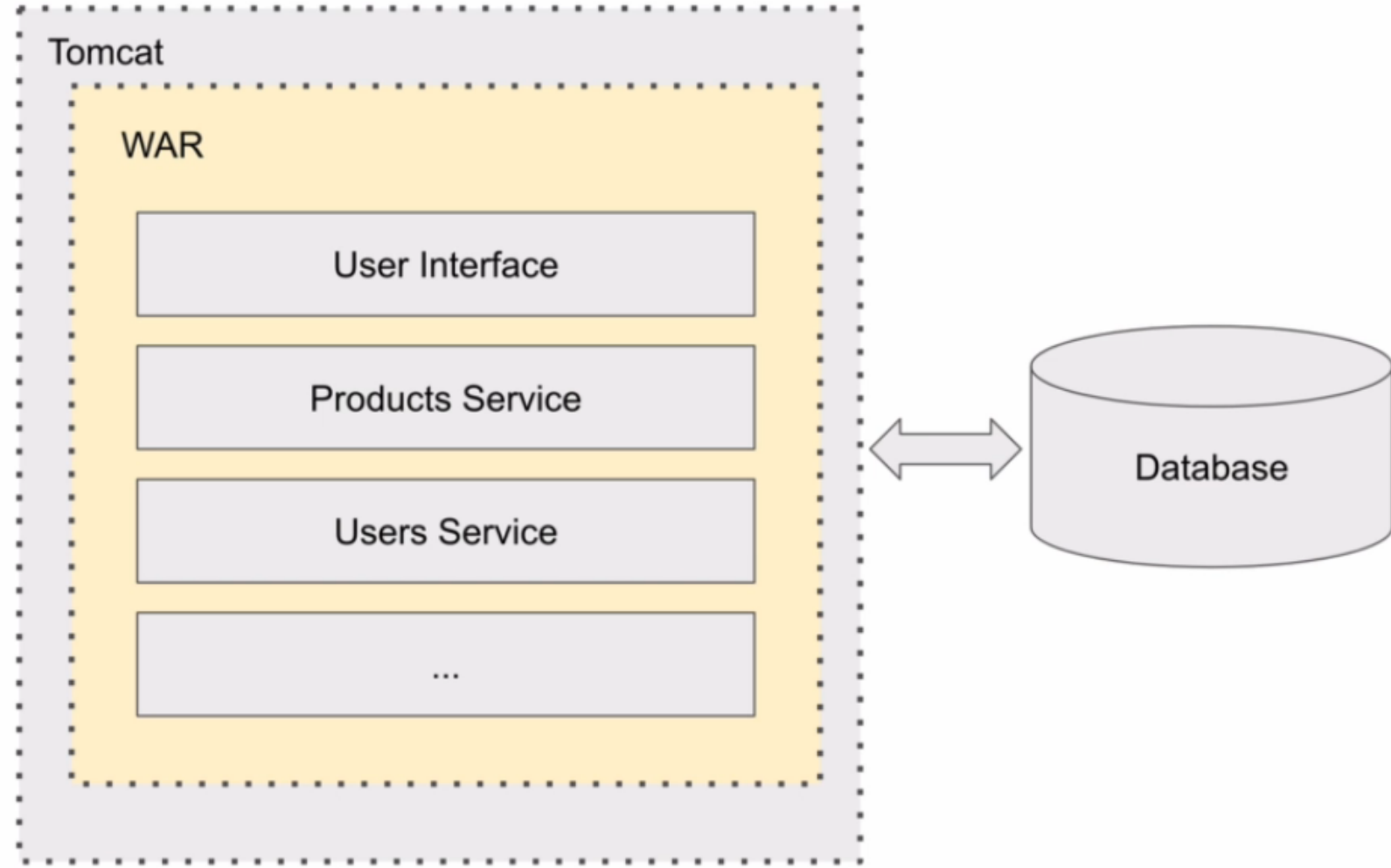


Day - 1

Microservices Architecture Overview

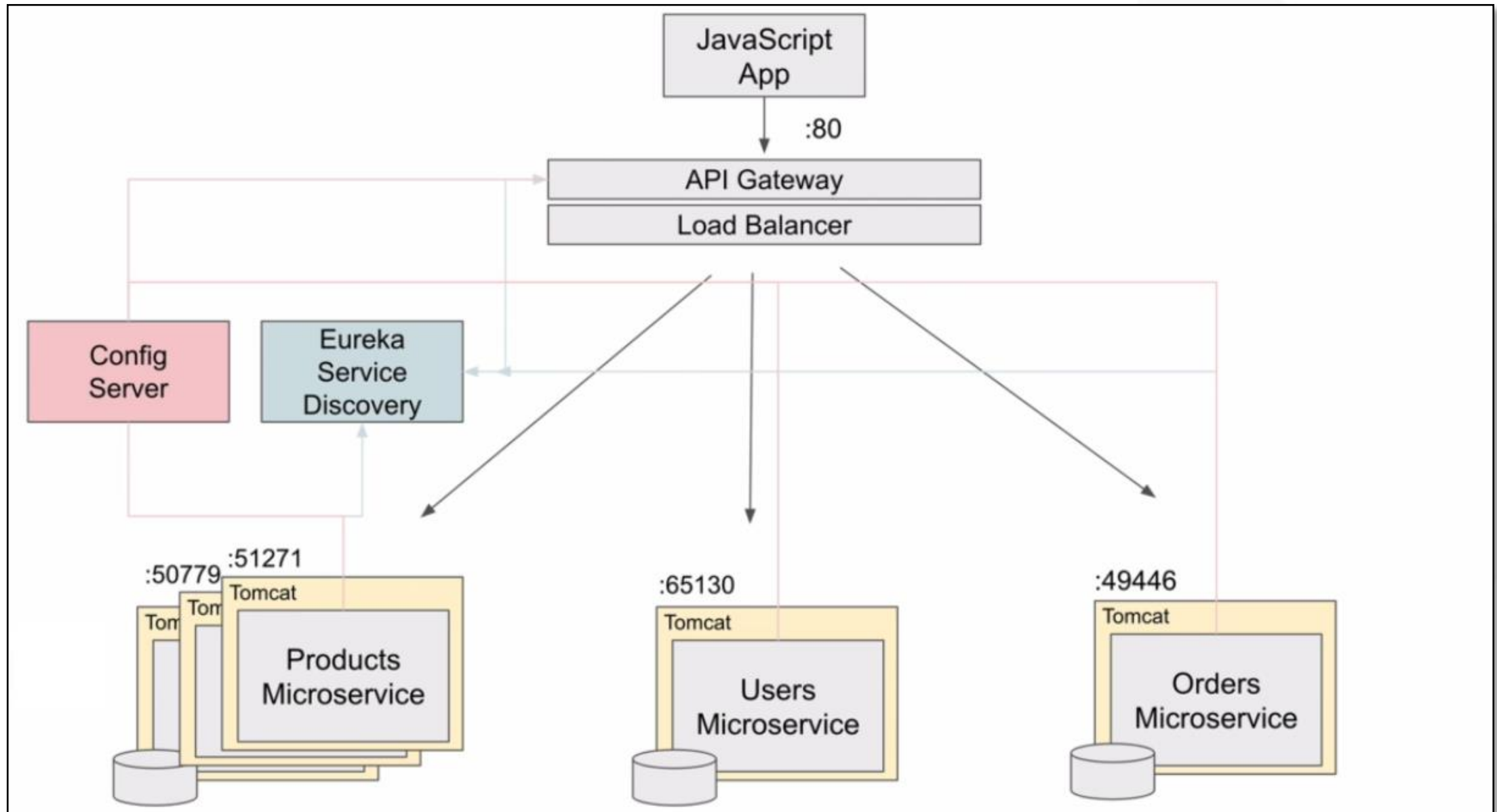


Monolithic Architecture



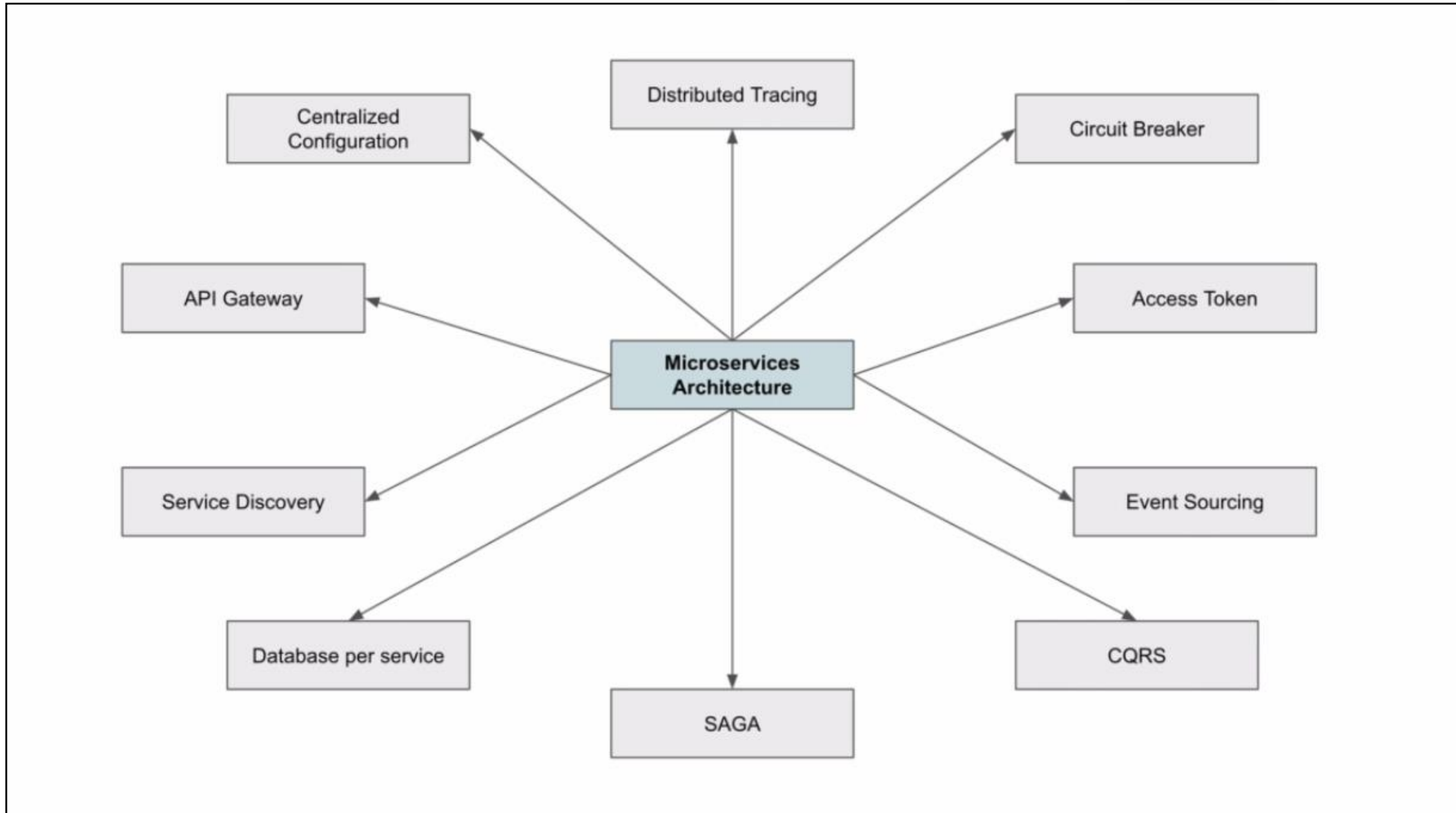


Microservices Architecture





Pattern Diagram





Day - 1

Event-Driven Microservices

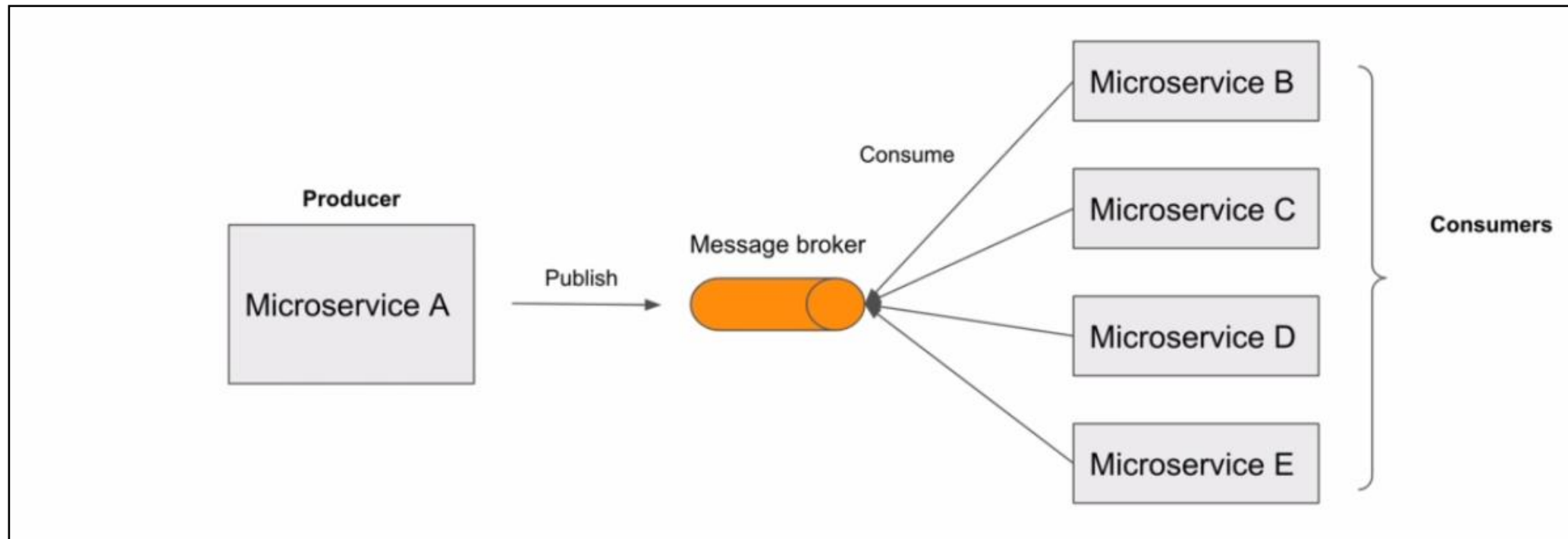
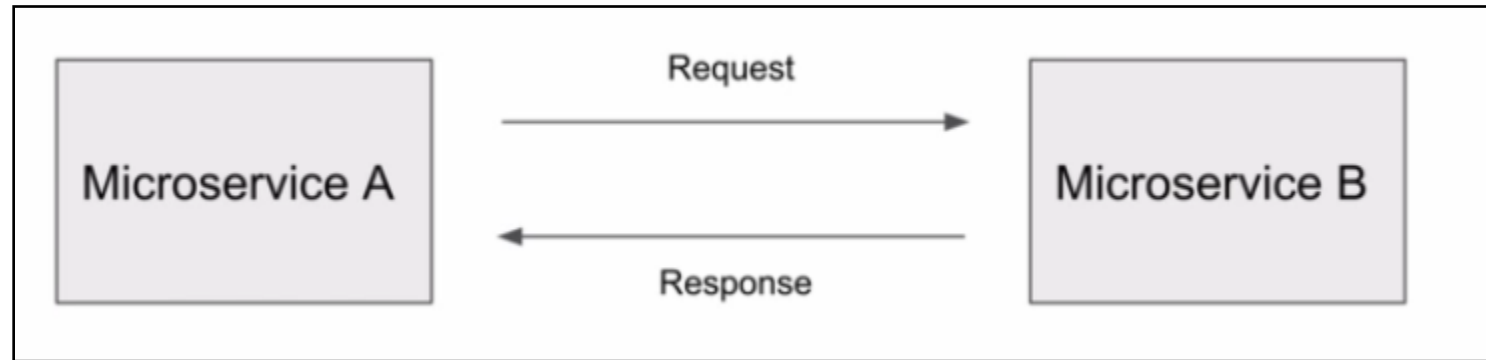


Event-Driven Microservices

- 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.

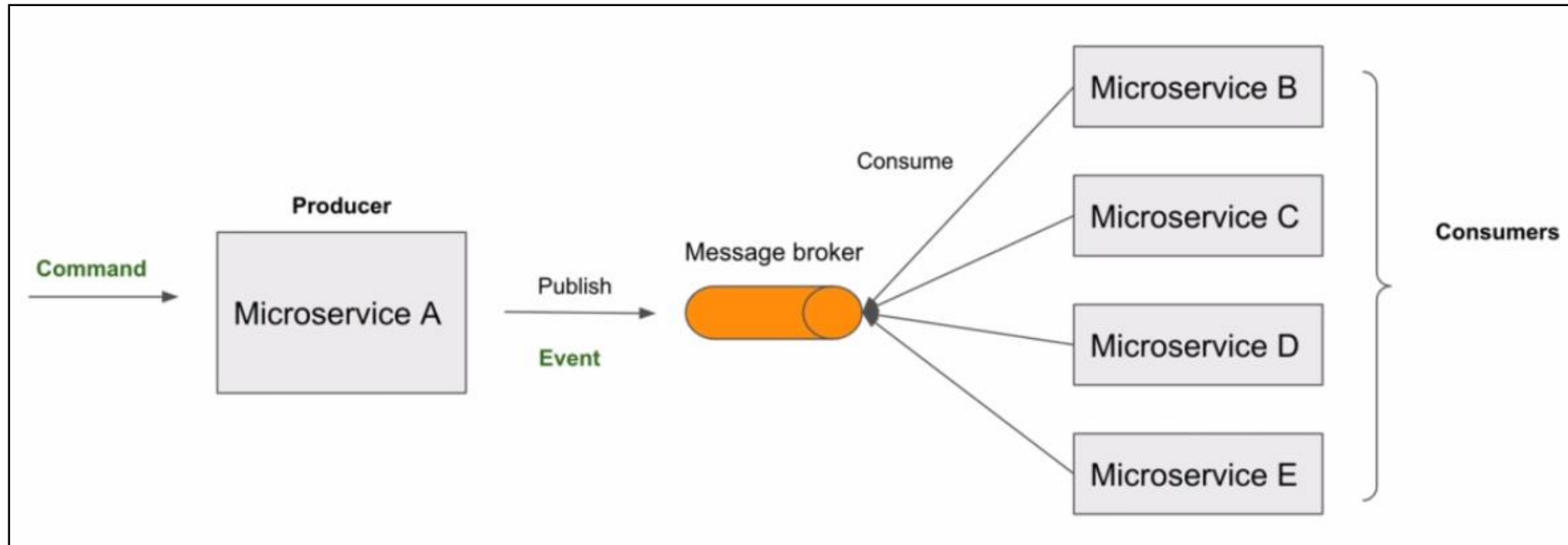
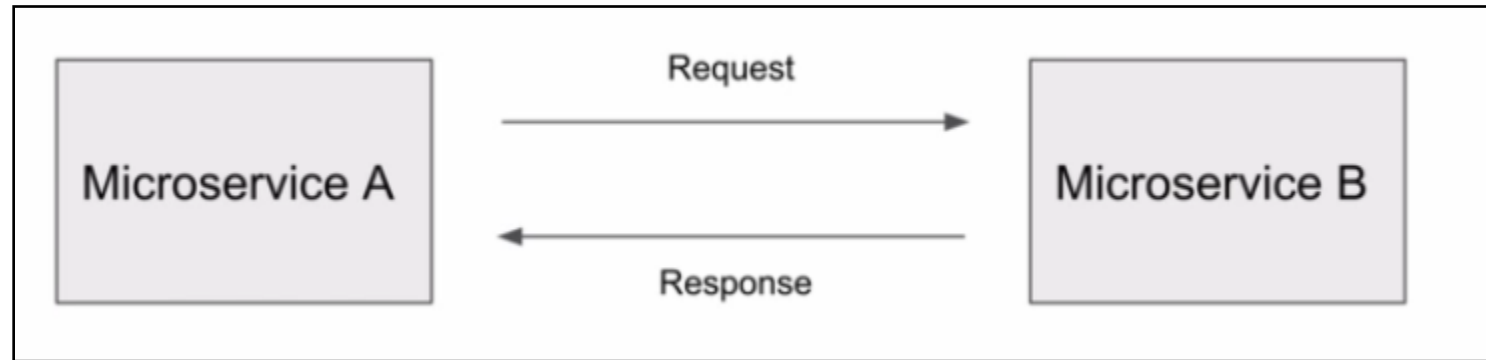


Event-Driven Microservices



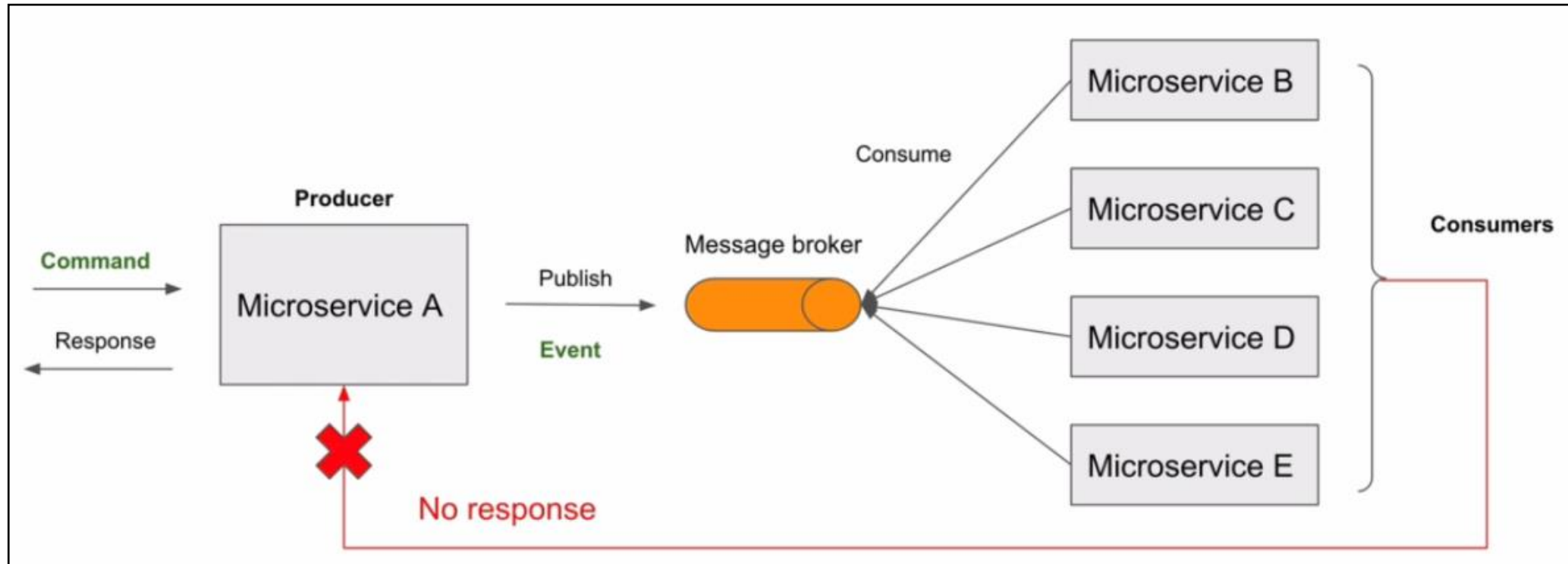
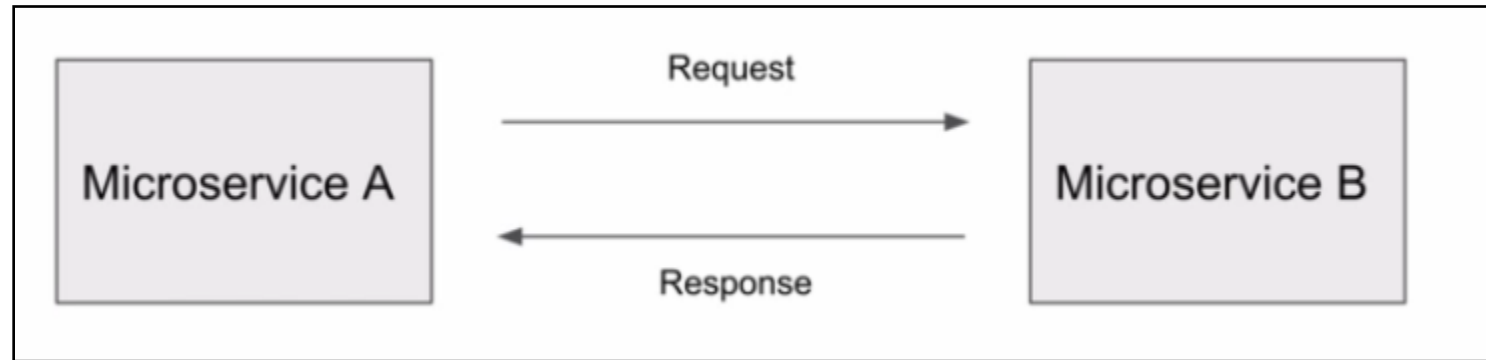


Event-Driven Microservices





Event-Driven Microservices





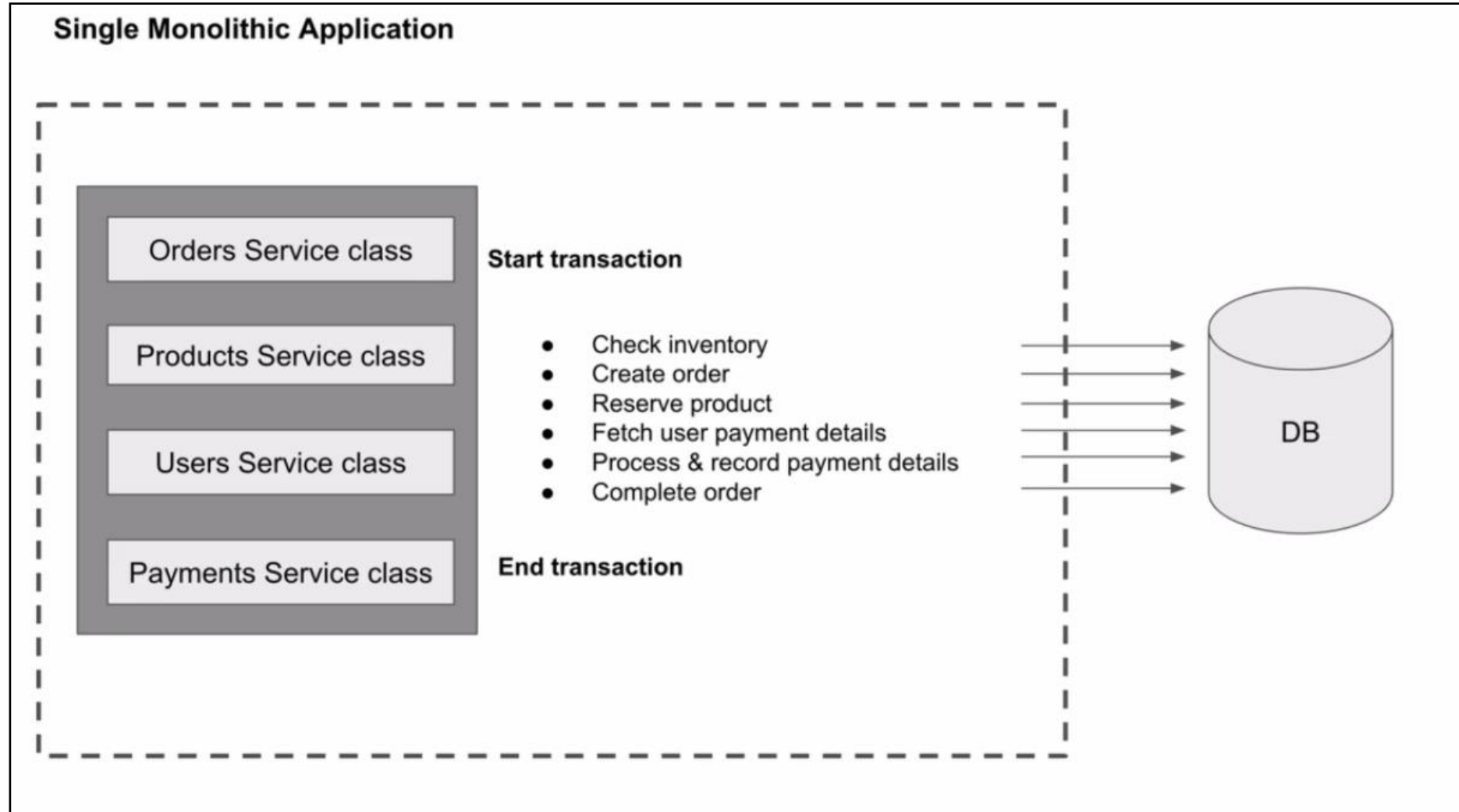
Day - 1

Transactions in Microservices



Transactions in Microservices

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



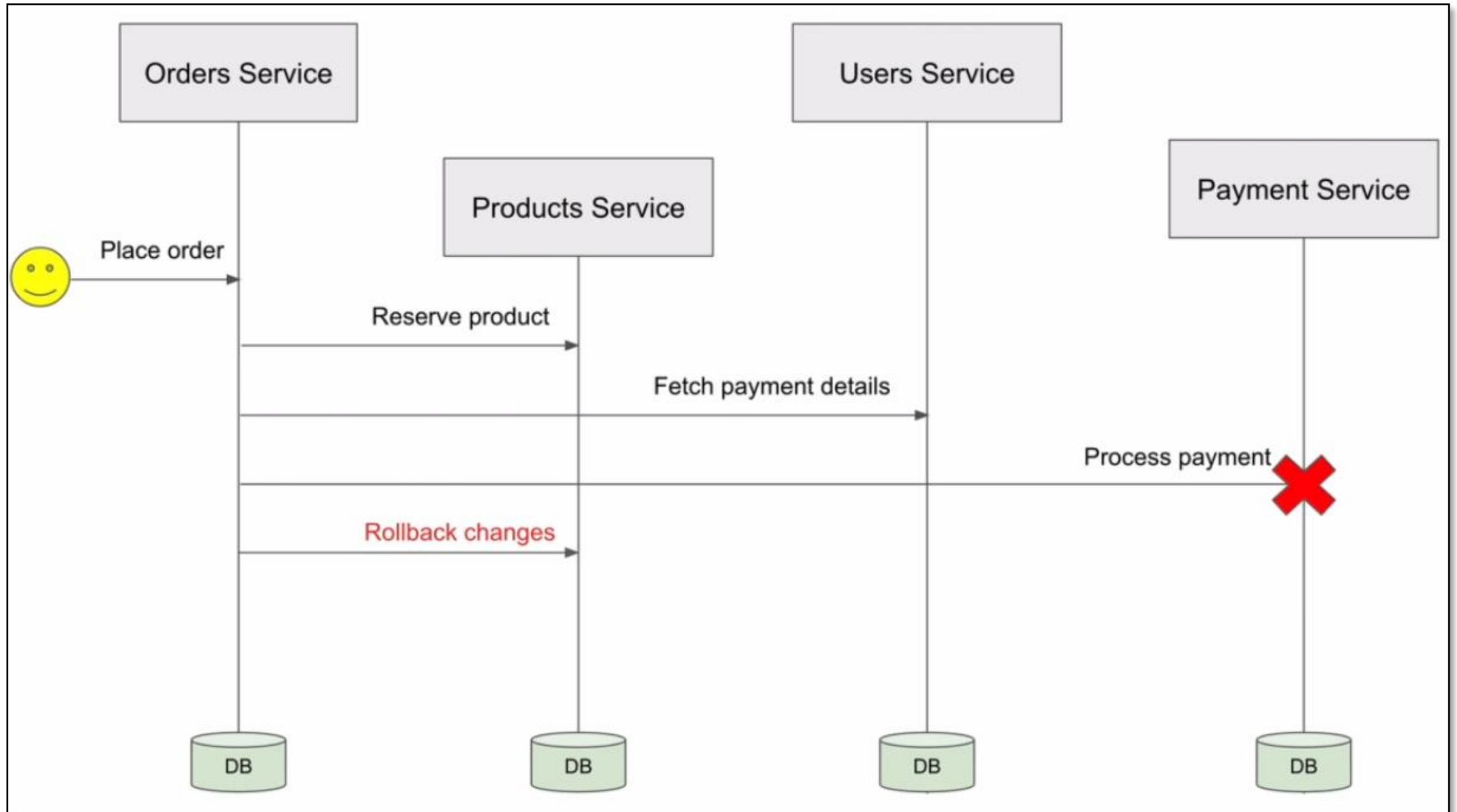


ACID Transactions

- 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

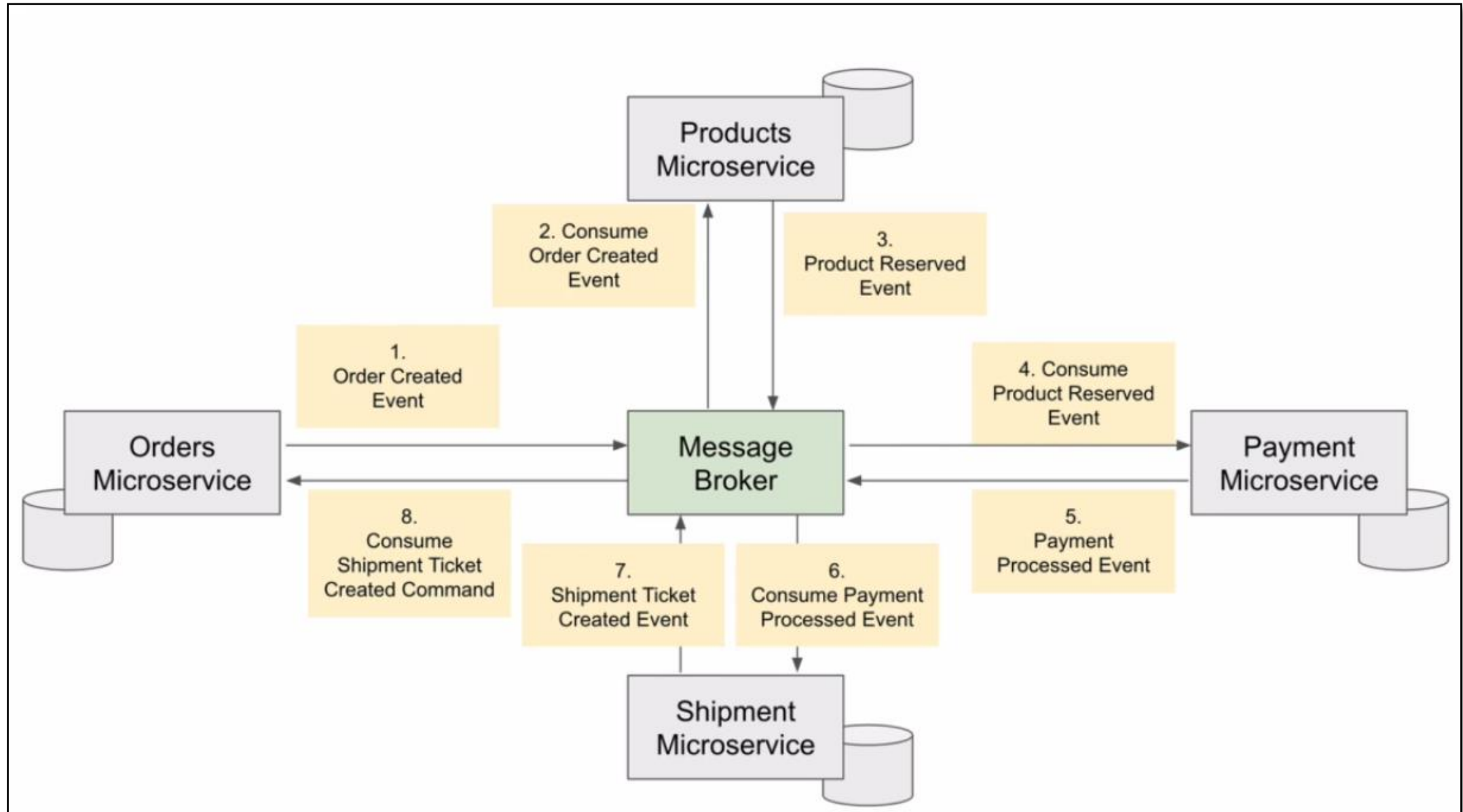


Choreography-Based SAGA

- 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



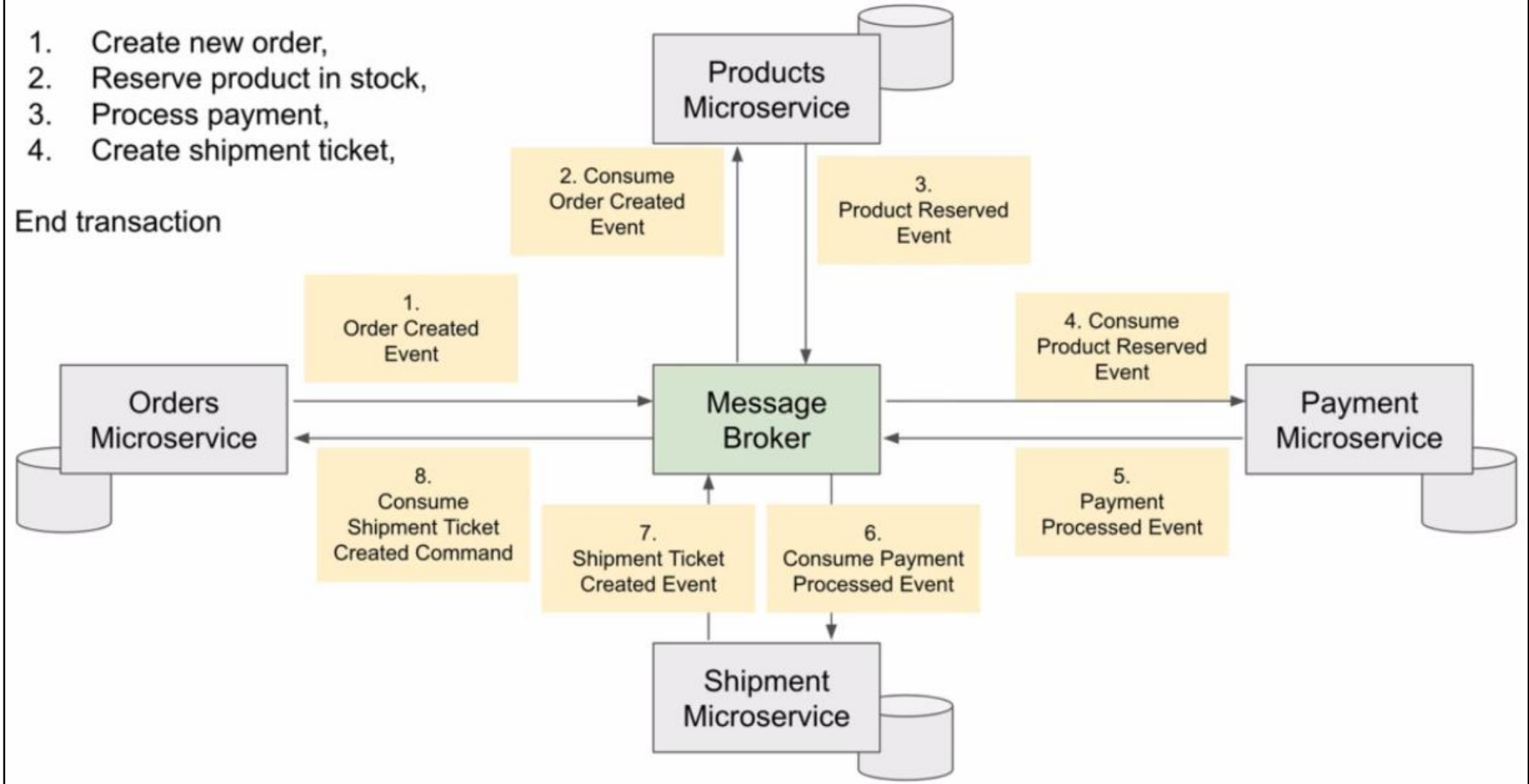


Choreography-Based SAGA

Begin Transaction

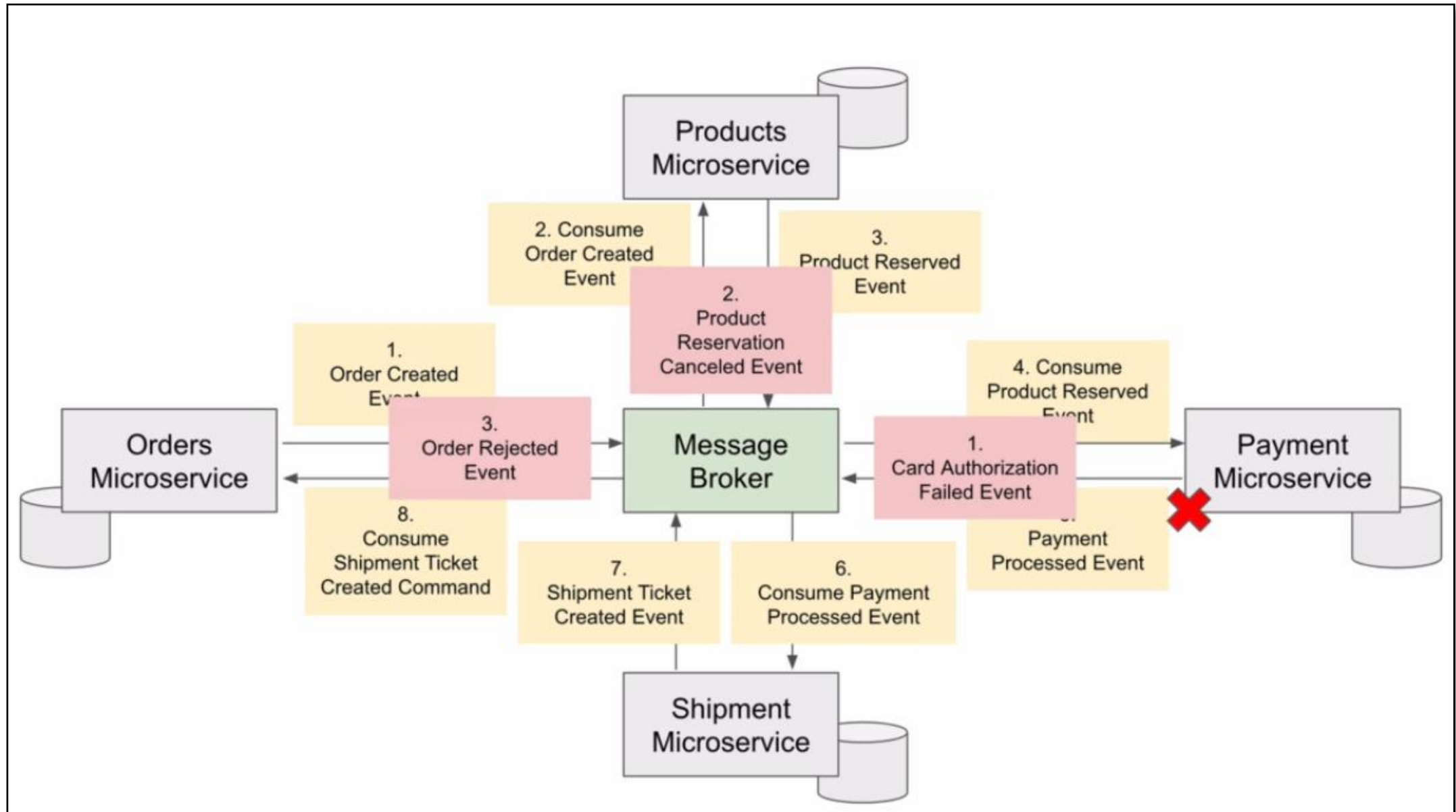
1. Create new order,
2. Reserve product in stock,
3. Process payment,
4. Create shipment ticket,

End transaction





Choreography-Based SAGA





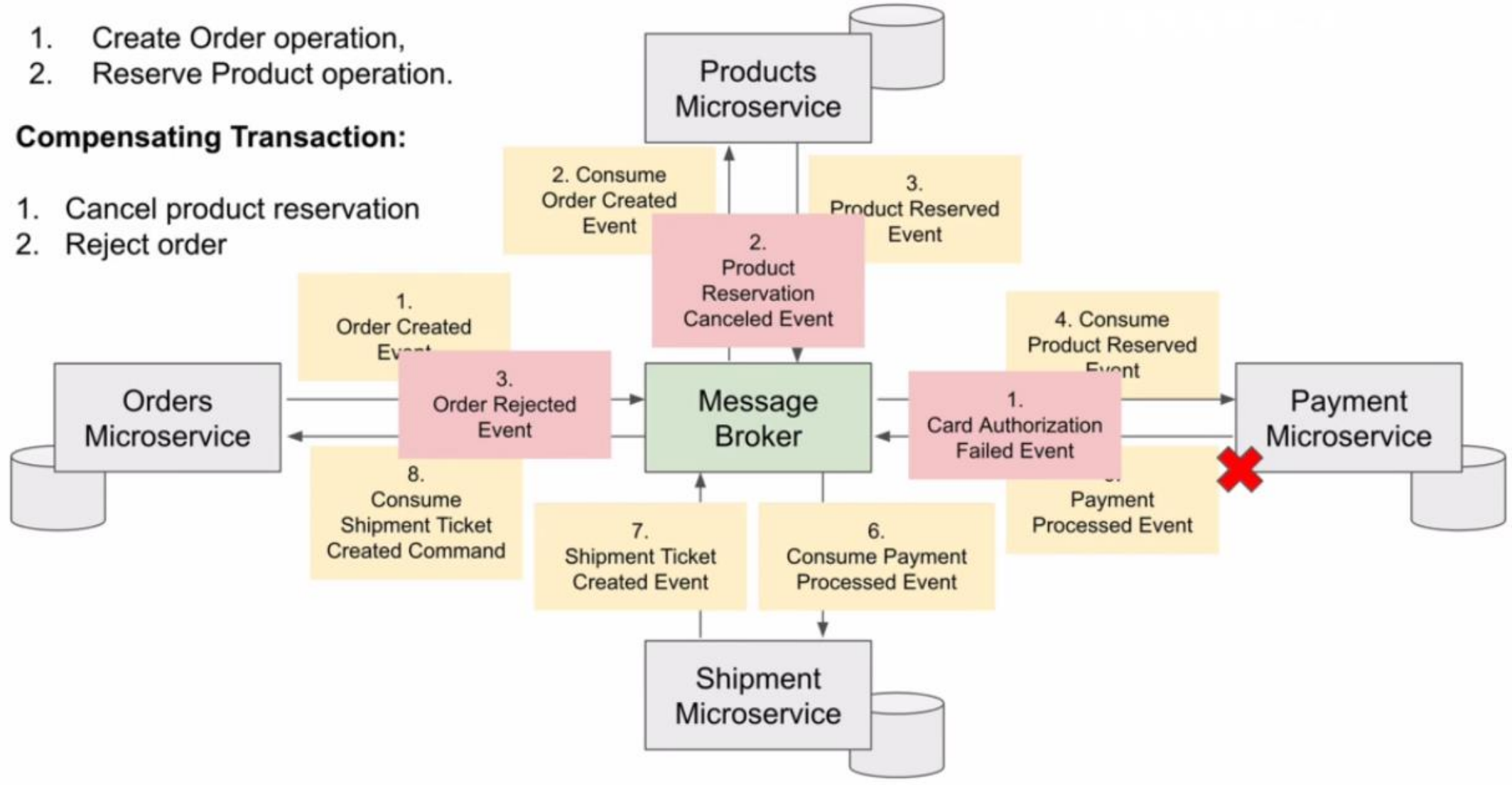
Choreography-Based SAGA

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:

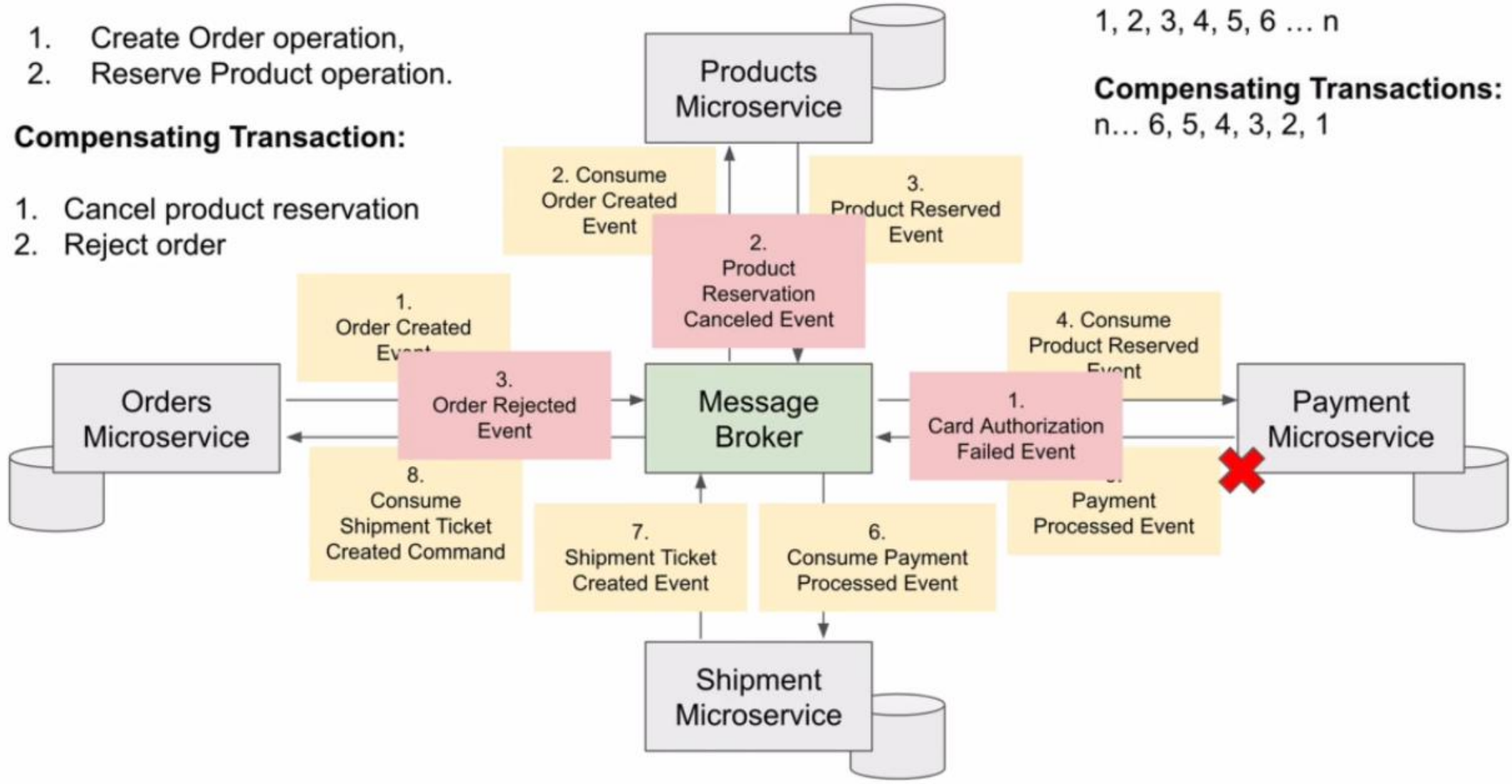
1. Cancel product reservation
2. Reject order

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

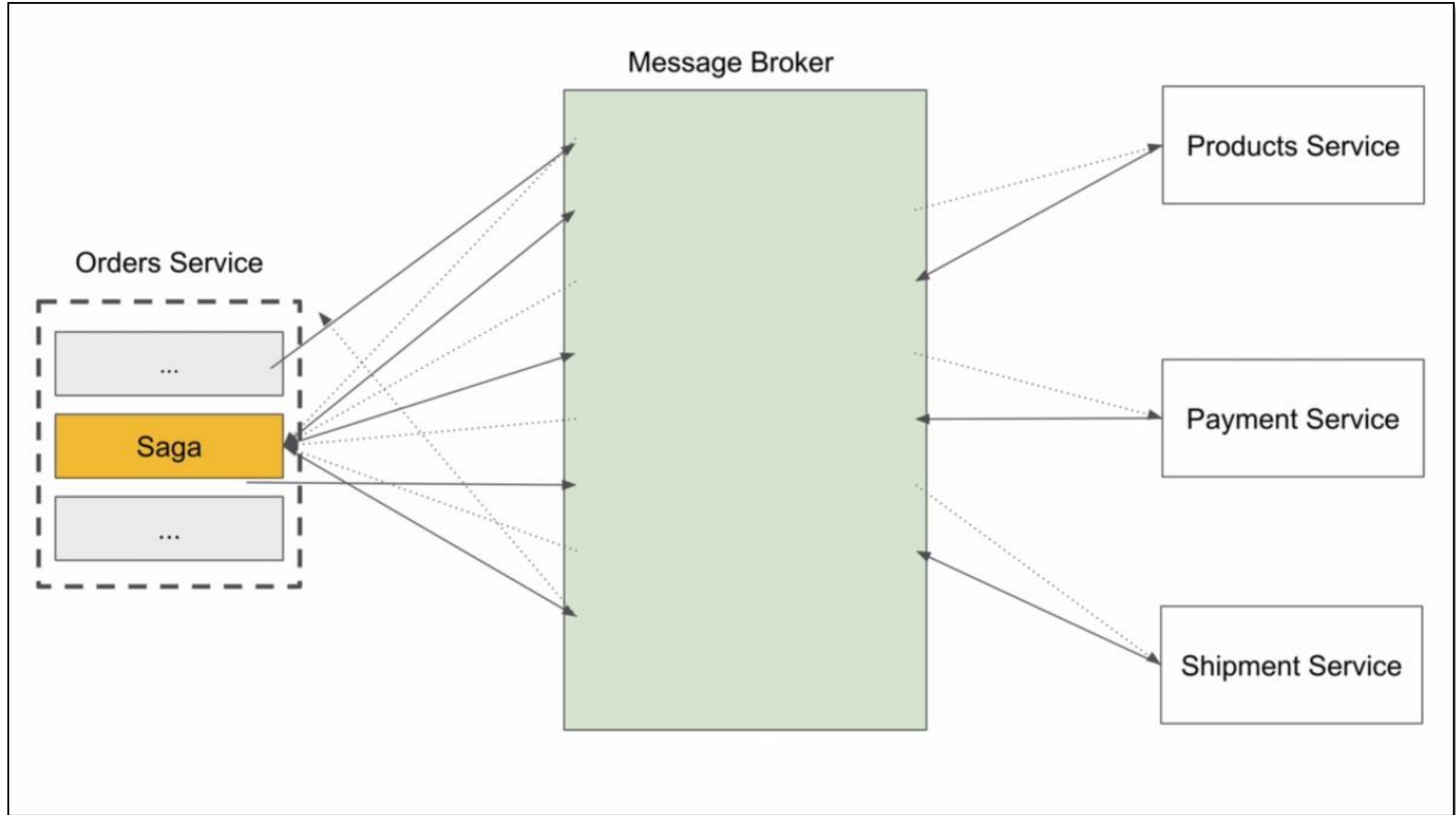


Orchestration-Based SAGA

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

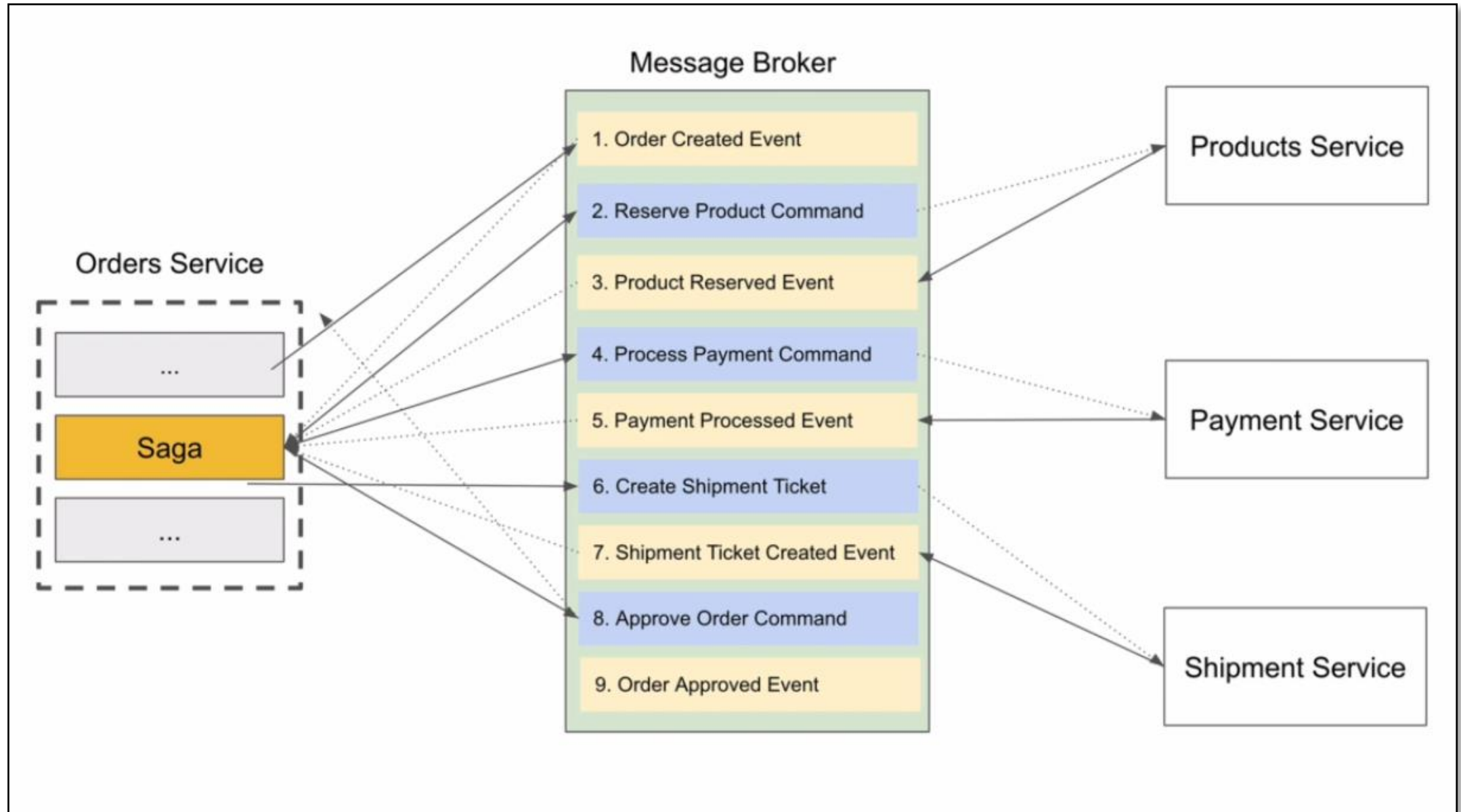


Orchestration-Based SAGA



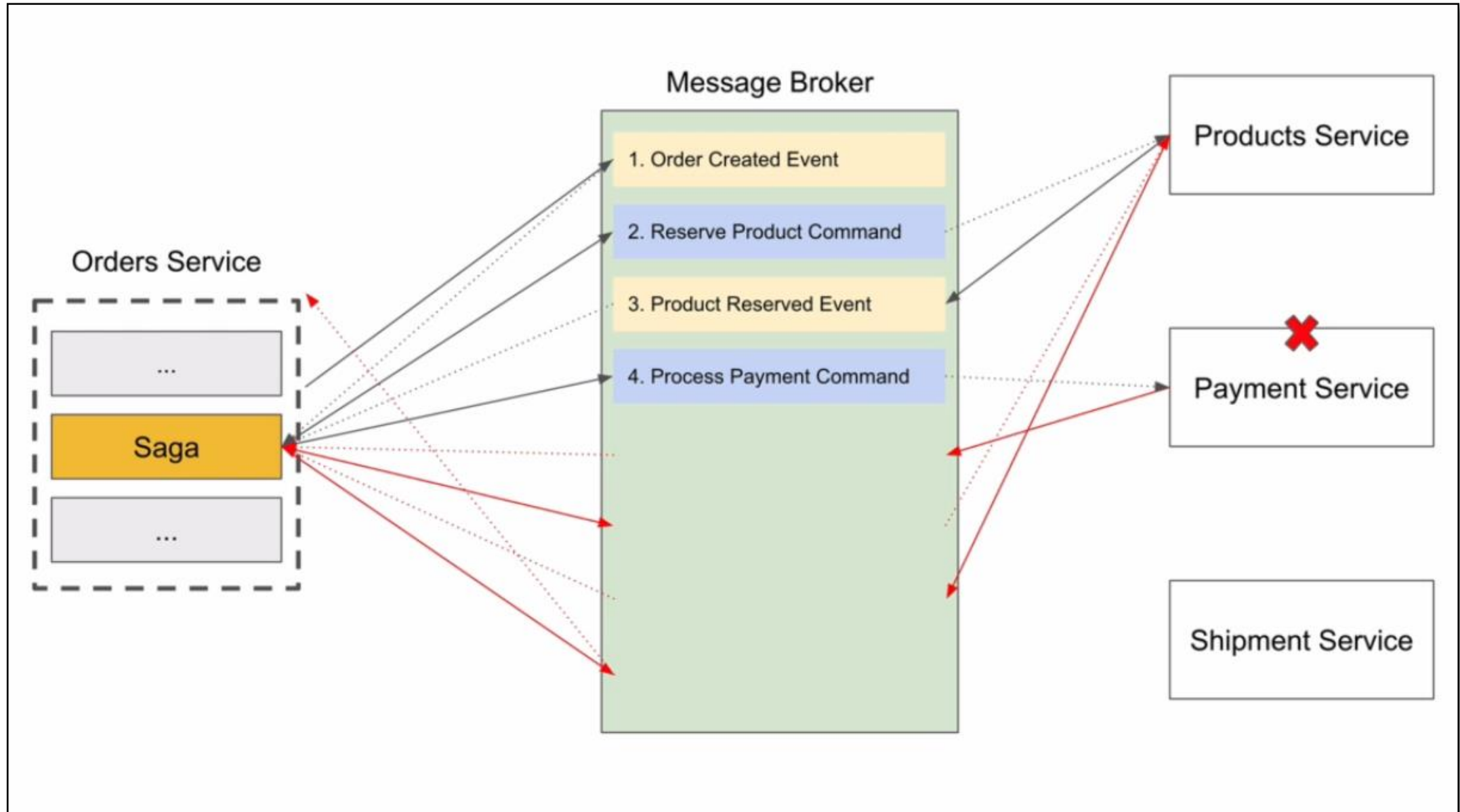


Orchestration-Based SAGA



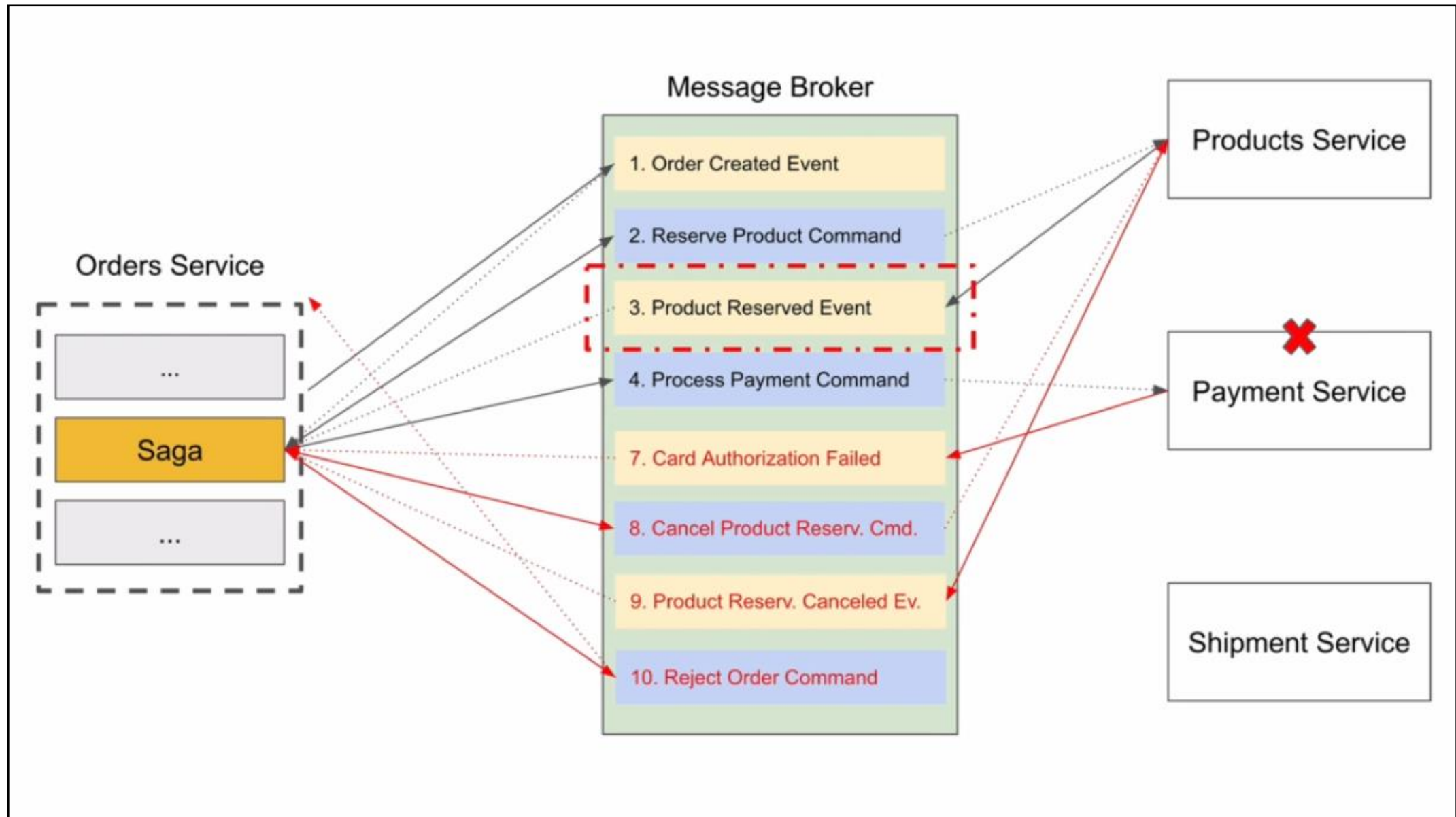


Orchestration-Based SAGA





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

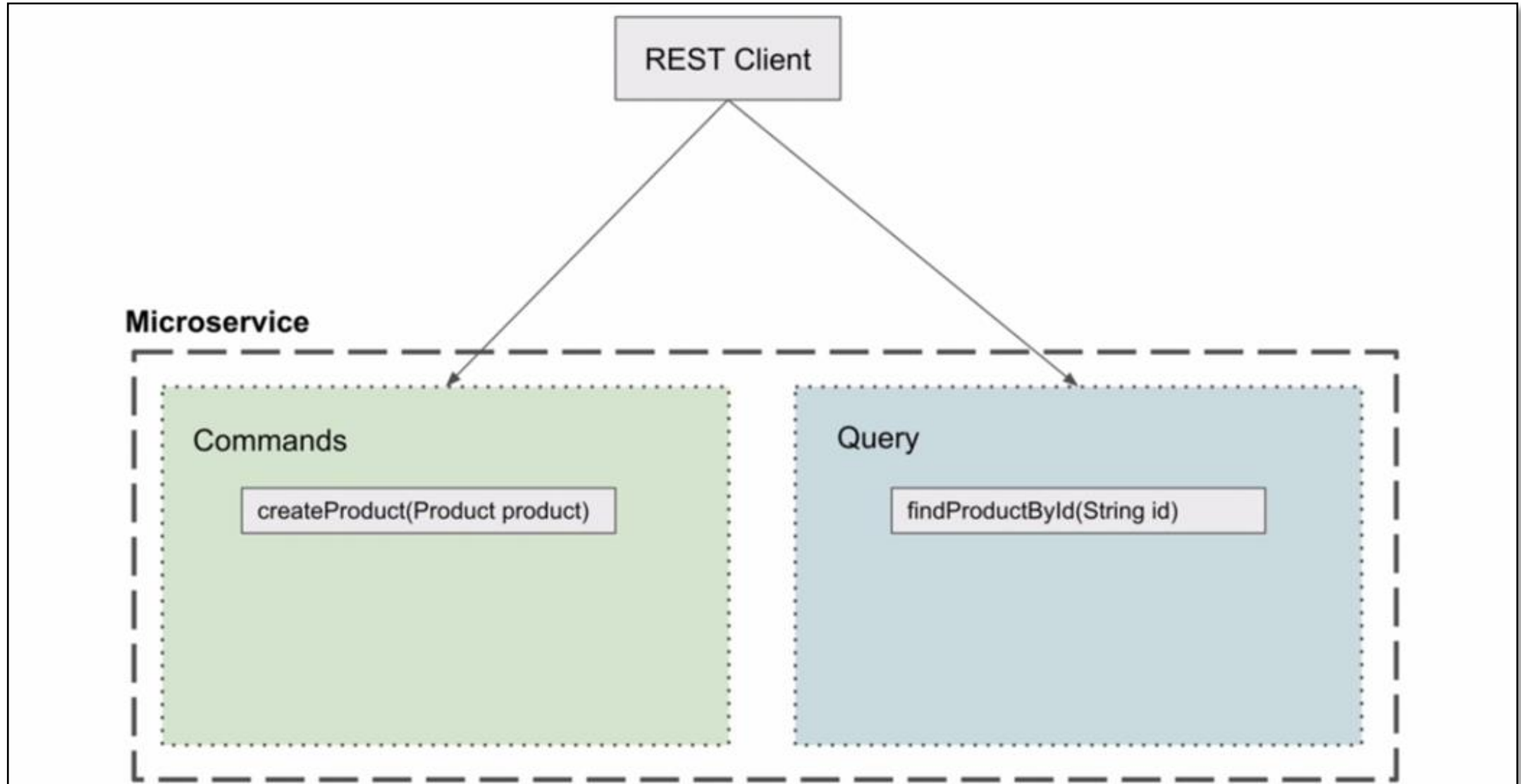


CQRS pattern

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

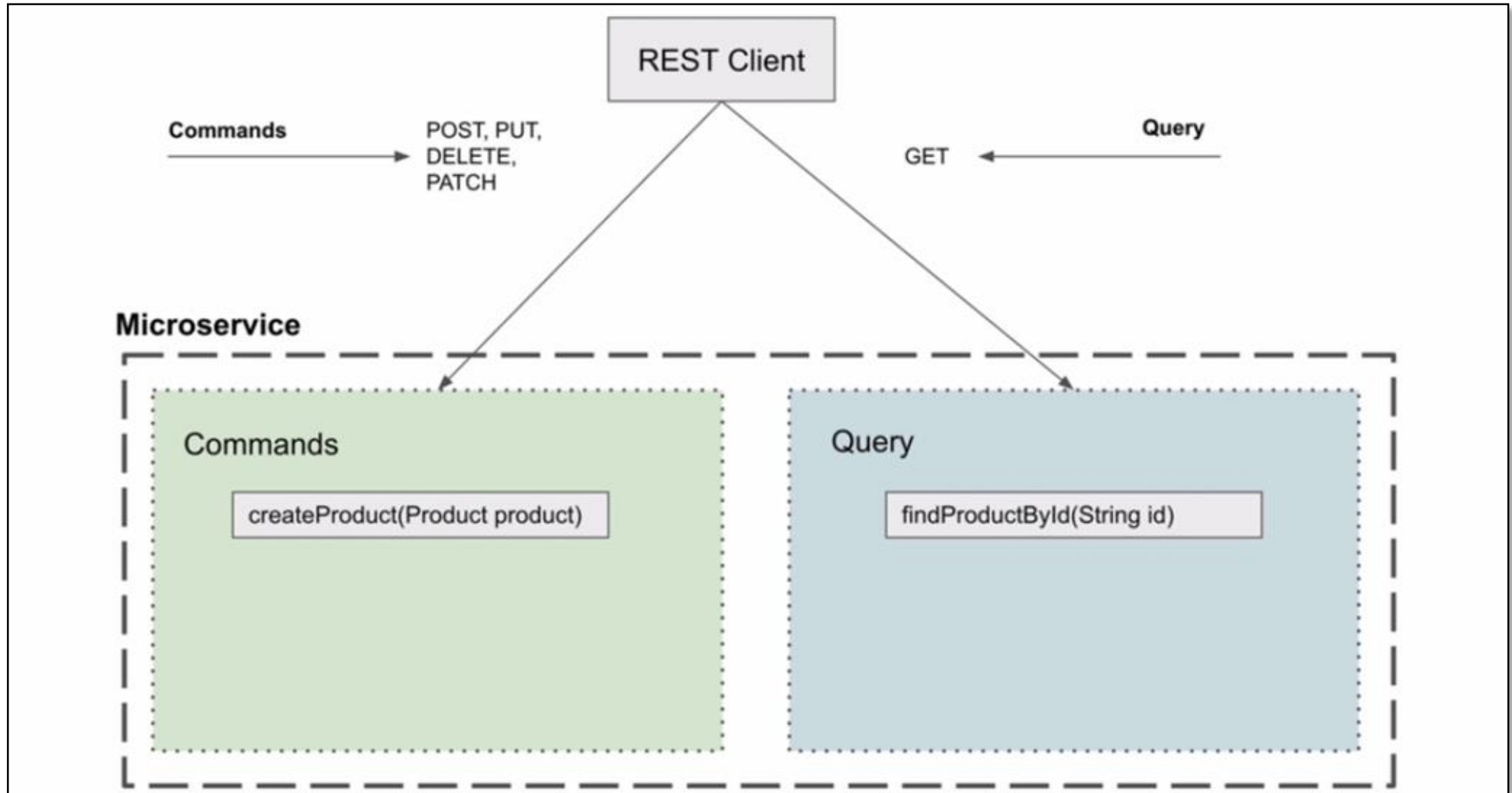


Command Query Responsibility Segregation



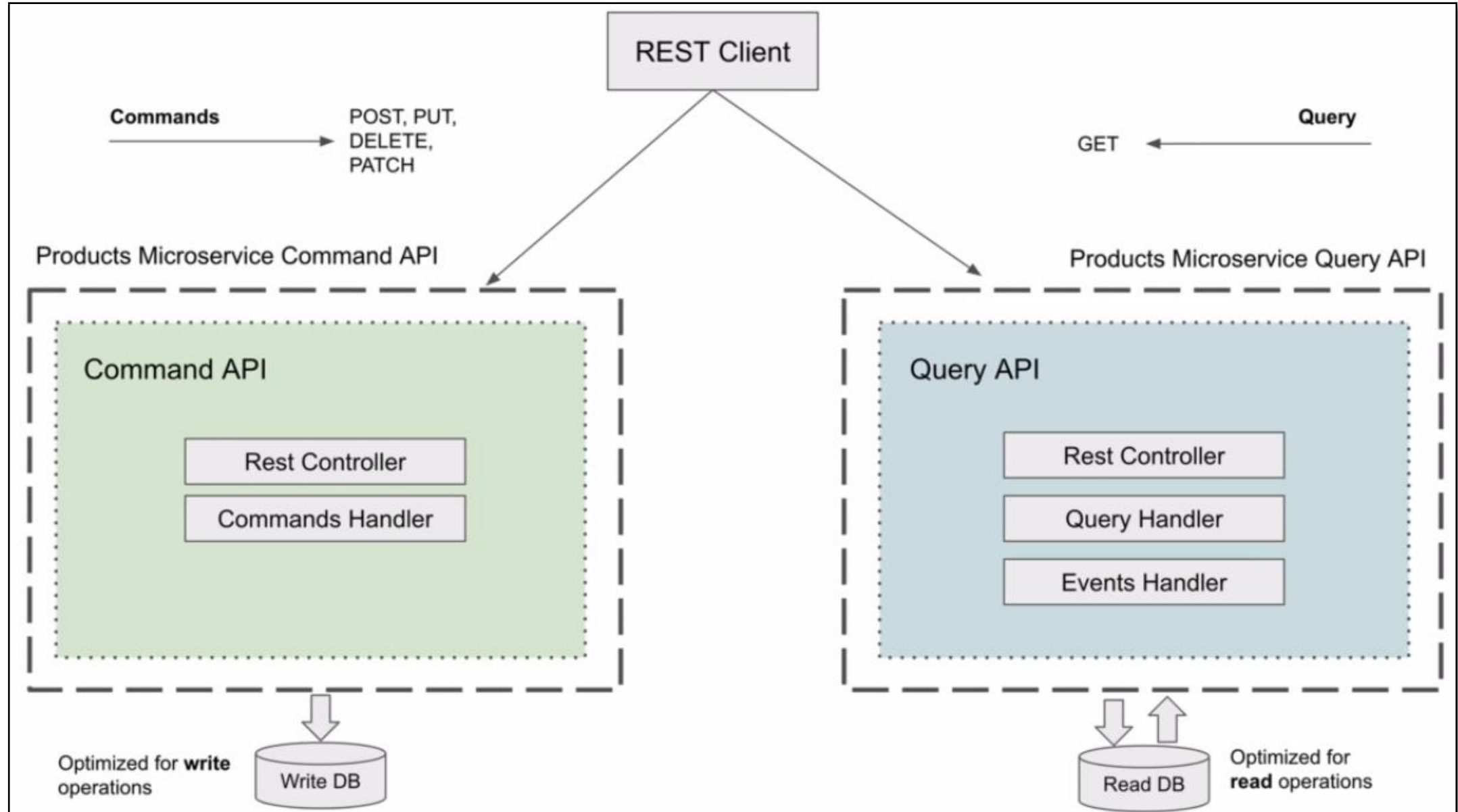


Command vs Query – Controllers



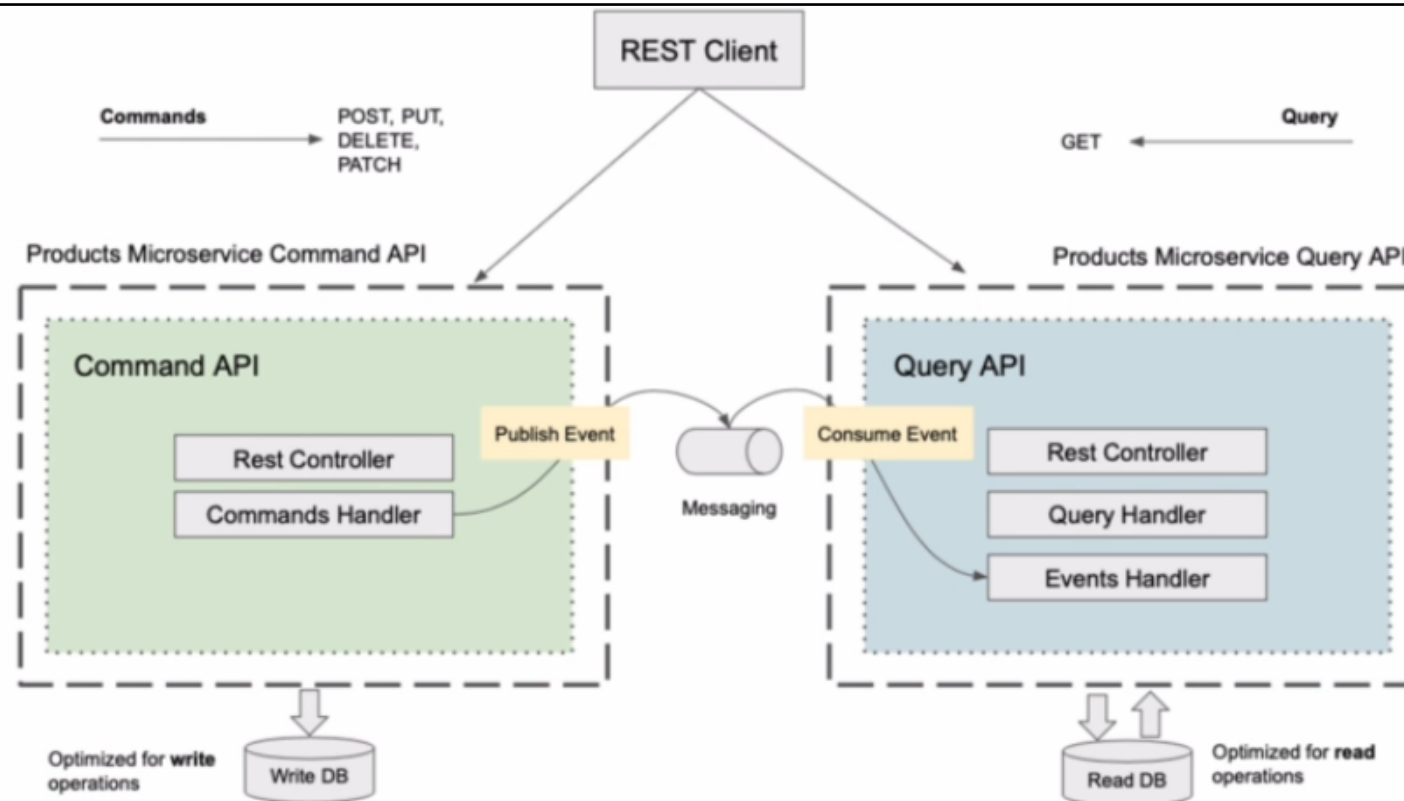


Read vs Write Traffic





Type of Messages



Type of Messages

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



Day - 1

CQRS and Event Sourcing

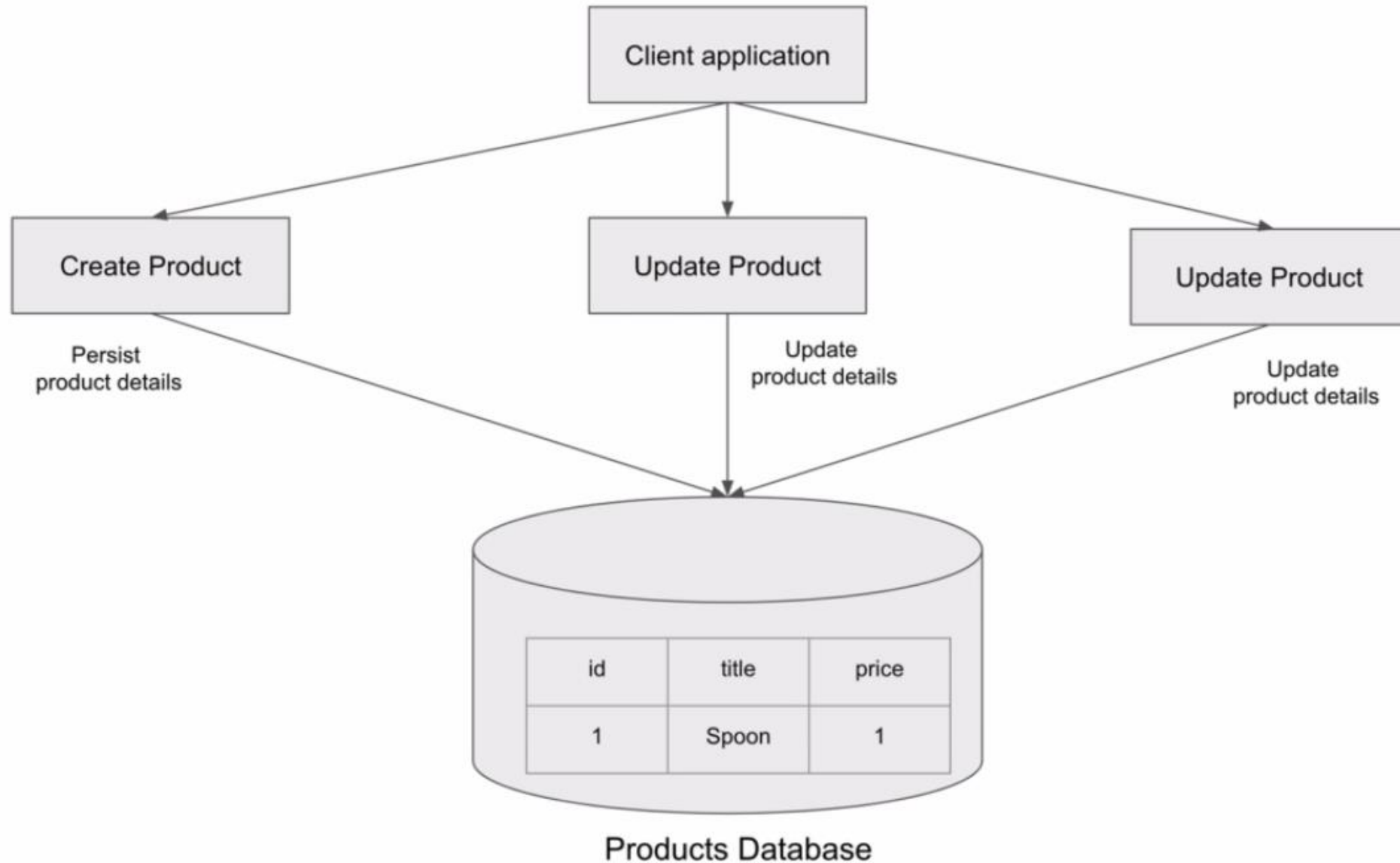


Event Sourcing

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

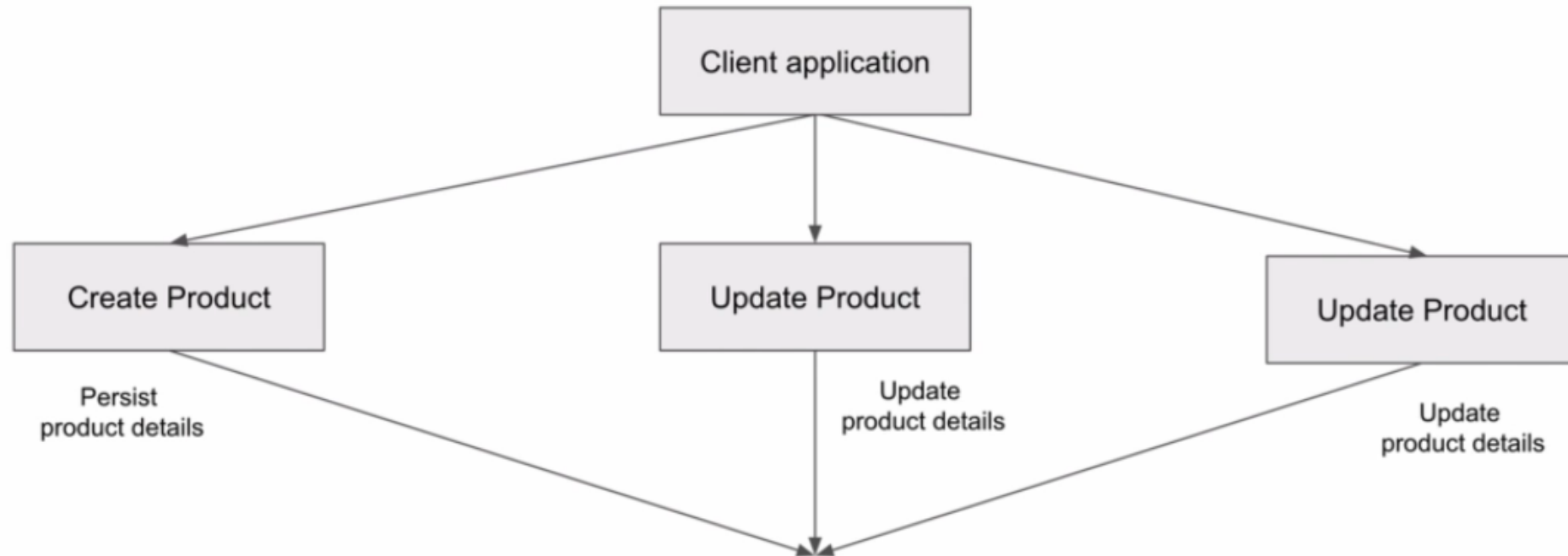


Event Sourcing





Event Sourcing

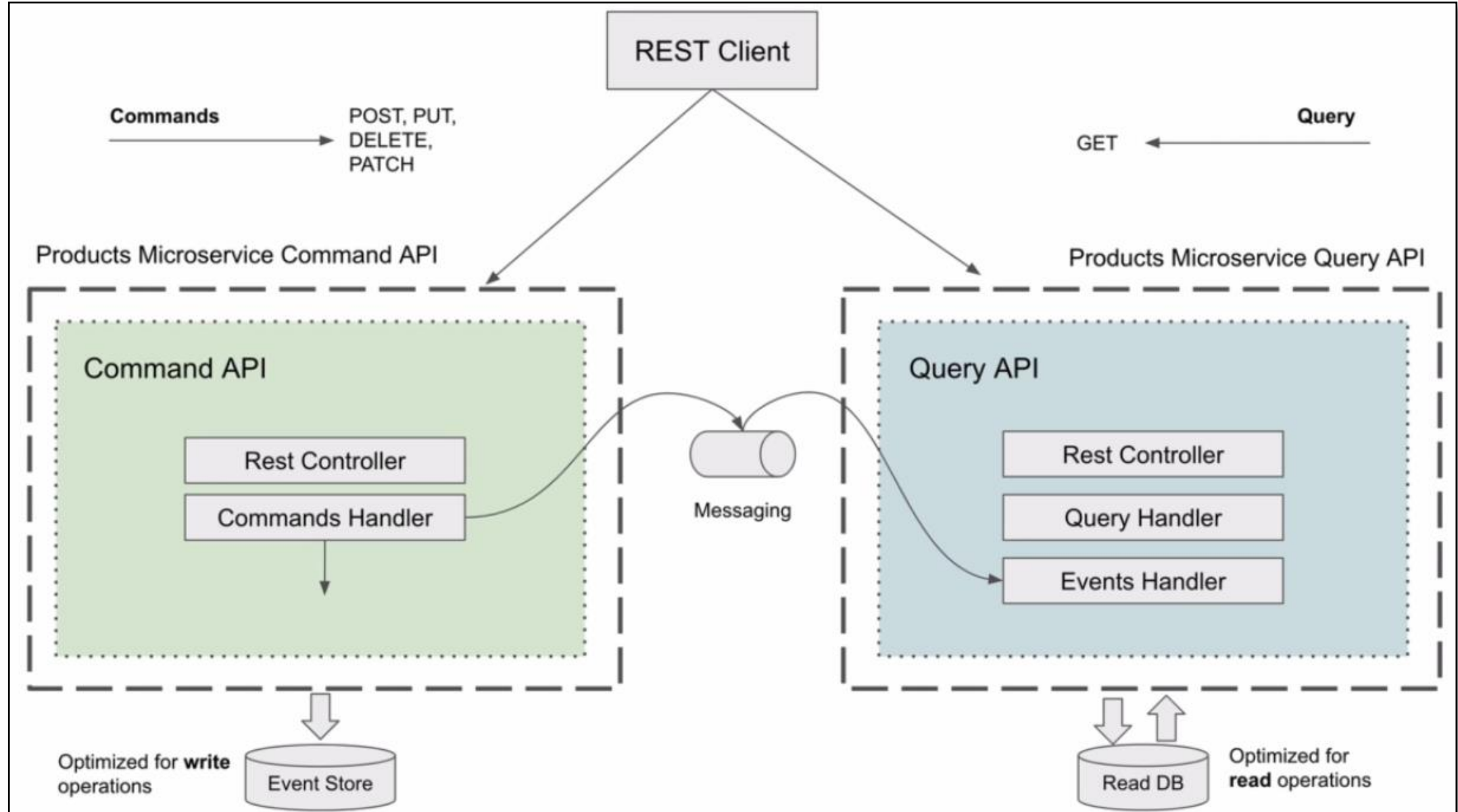


eventIdentifier	eventType	eventPayload
5e584f4e	ProductCreatedEvent	{"productId":"4ee43c56","title":"Spoon", "price":1}
60e74f5b	ProductUpdatedEvent	{"productId":"4ee43c56","title":"Spoon", "price":2}
d6f2d6c5	ProductUpdatedEvent	{"productId":"4ee43c56","title":"Spoon", "price":3}

Event Store

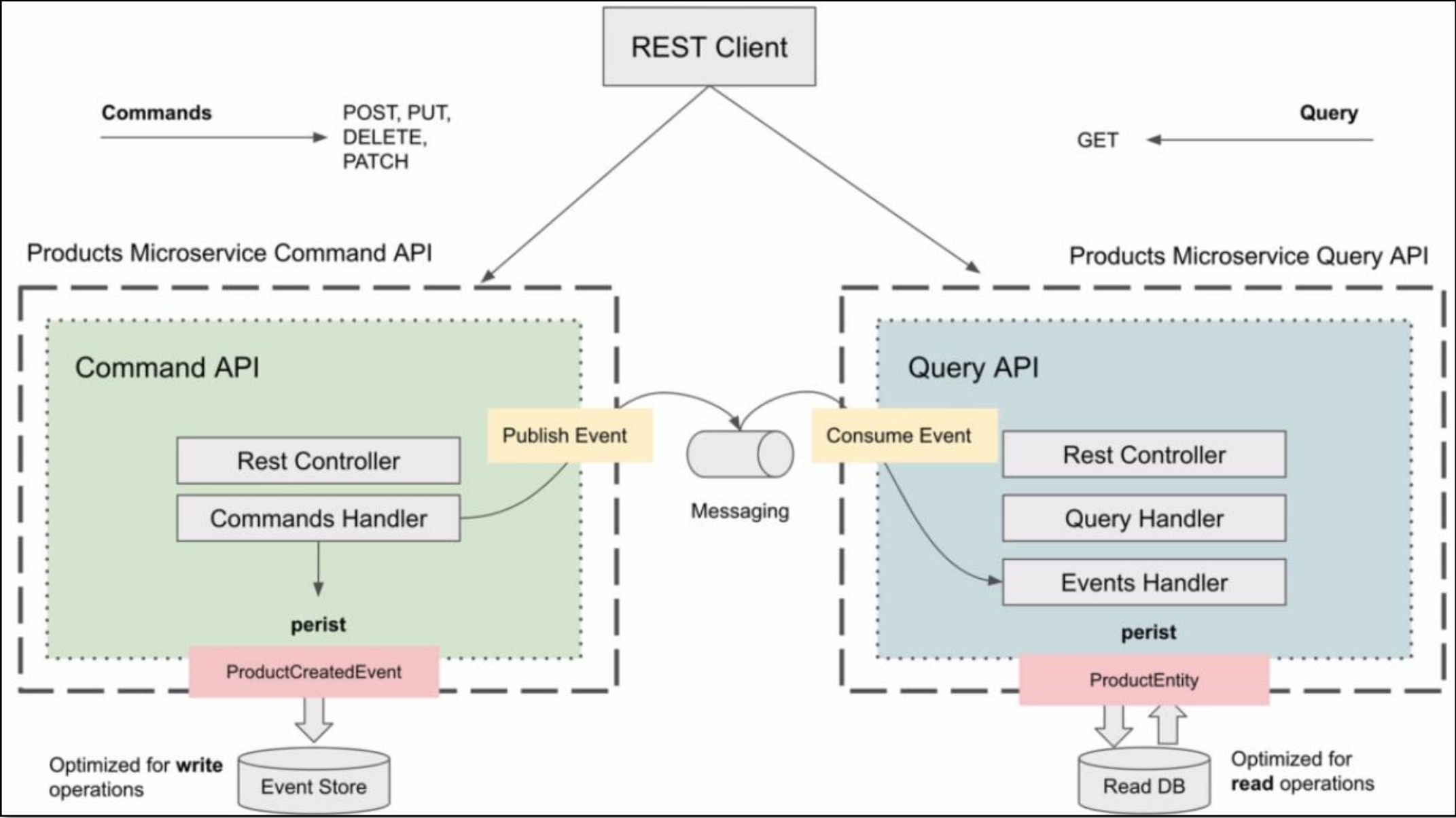


CQRS and Event Sourcing





CQRS and Event Sourcing





CQRS and Event Sourcing

Event Store

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

Read Database

id	title	price
1	Spoon	3



CQRS and Event Sourcing

Event Store

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

Replay



Read Database

id	title	price
1	Spoon	3

Read Database 2

title	price
Spoon	3



Day - 1

Building Microservices with Spring Boot



Building a RESTful web application

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

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



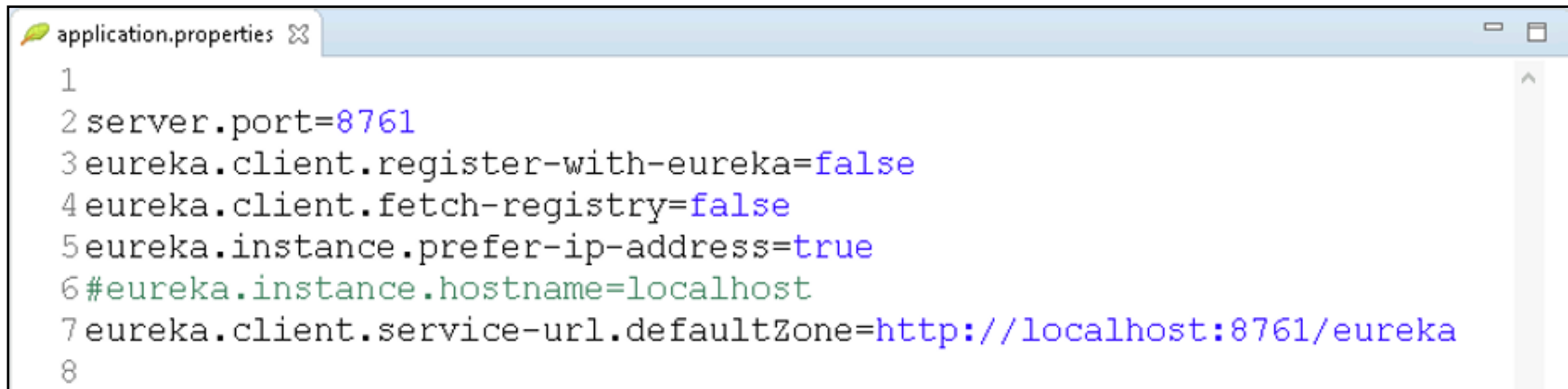
Configure Microservices with Eureka Service Registry Server

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



Configure Microservices with Eureka Service Registry Server

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



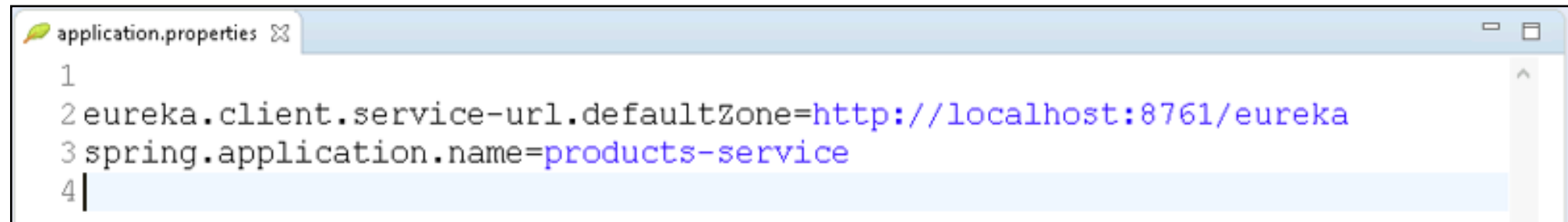
```
1
2 server.port=8761
3 eureka.client.register-with-eureka=false
4 eureka.client.fetch-registry=false
5 eureka.instance.prefer-ip-address=true
6 #eureka.instance.hostname=localhost
7 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
8
```

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.



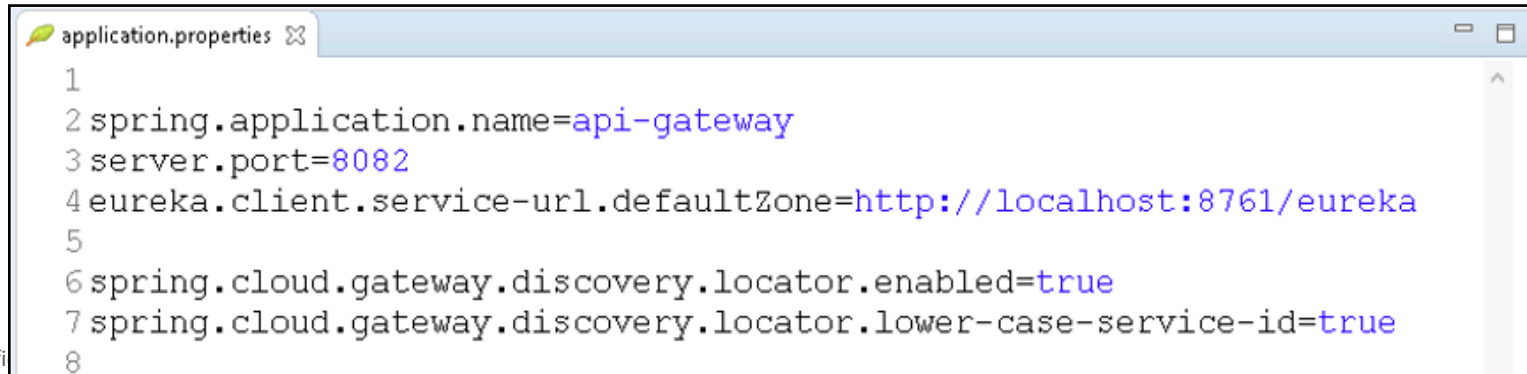
```
application.properties
1
2 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
3 spring.application.name=products-service
4 |
```

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



Implementing Spring Cloud Gateway in Microservices

1. Ensure the Discovery Server and Product Service is running.
2. Now let's implement an API gateway that acts as a single-entry point for a collection of microservices.
3. Create a new Project for **ApiGateway** using the Spring 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.

A screenshot of a code editor window titled 'application.properties'. The window shows a list of 8 lines of configuration. Line 2 sets 'spring.application.name' to 'api-gateway'. Line 3 sets 'server.port' to '8082'. Line 4 sets 'eureka.client.service-url.defaultZone' to 'http://localhost:8761/eureka'. Line 6 sets 'spring.cloud.gateway.discovery.locator.enabled' to 'true'. Line 7 sets 'spring.cloud.gateway.discovery.locator.lower-case-service-id' to 'true'.

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



Implementing Spring Cloud Gateway in Microservices

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





Recap of Day – 1

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



Day - 2



Day – 2 Agenda

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



Day – 2 Agenda

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



Day - 2

Axon – Getting Started



- Axon 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.



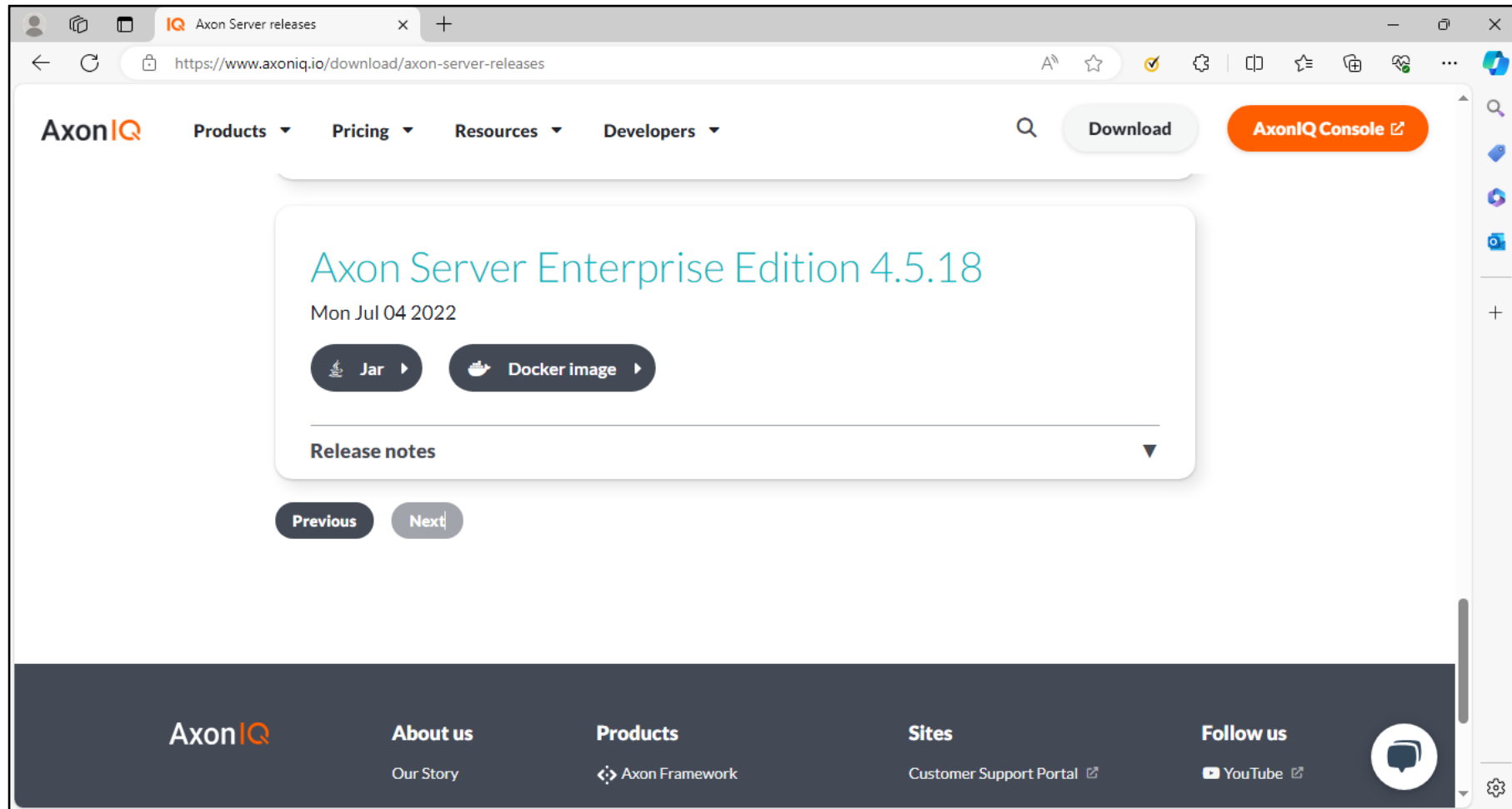
Day - 2

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>

AxonDashboard: settings x +

localhost:8024

Axon Server

SSL disabled

Authentication disabled

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

Edition Standard Edition

6:39 AM 7/4/2023



Day - 2

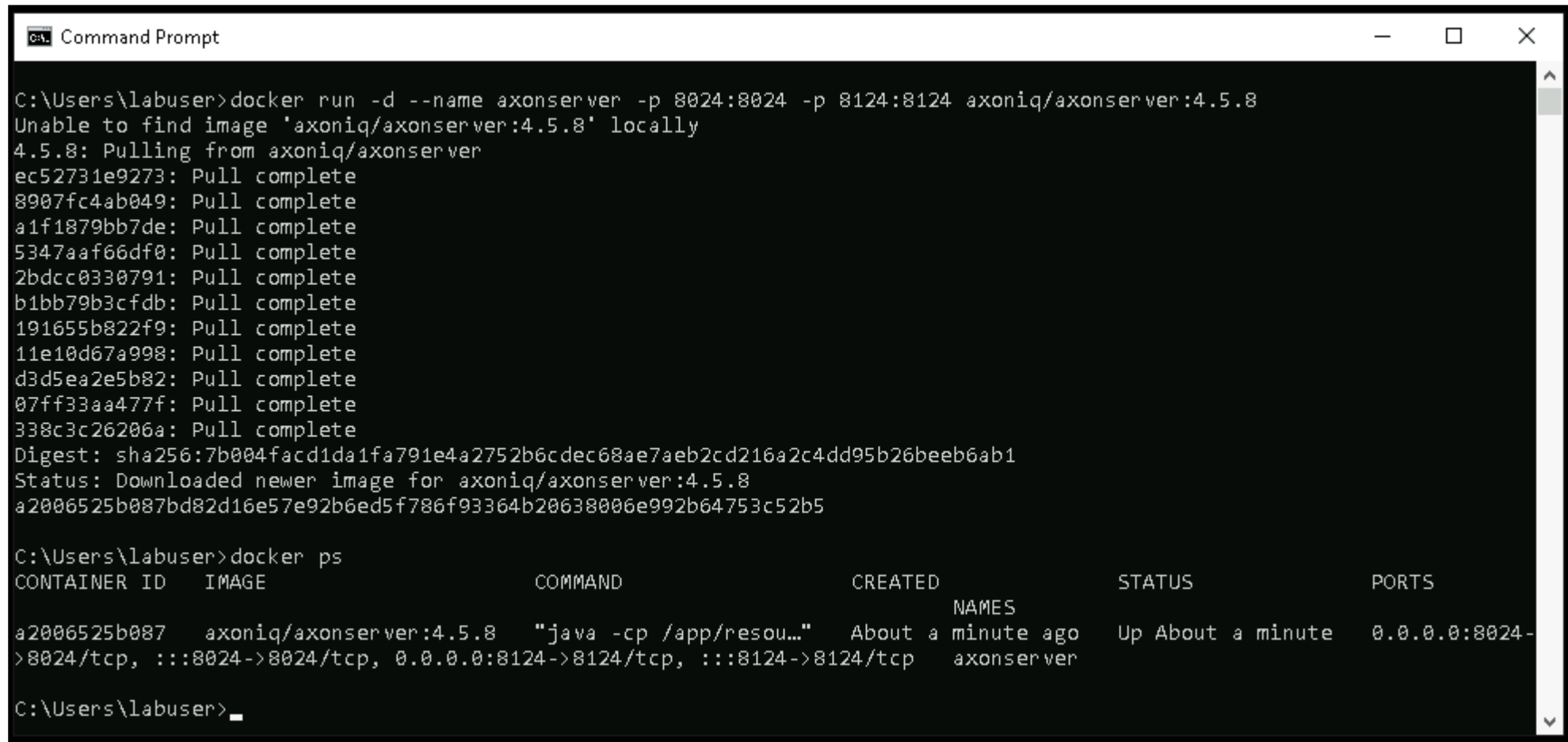
Run Axon Server in a Docker Container



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`



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

C:\Users\labuser>docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
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
axonserv
```



Start, Stop, Delete Axon Server Docker Container By ID

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



Day - 2

Bringing CQRS and Event Sourcing Together with Axon Framework



Introduction

- Apply the CQRS design pattern to our Products Microservices.



Accept HTTP Request Body

- Add the Lombok dependency:

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



Accept HTTP Request Body

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

```
CreateProductRestModel.java
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:

```
ProductController.java
1 package com.mphasis.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5
6
7
8
9
10
11
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:

```
CreateProductCommand.java
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 }
21
```



Send Command to a Command Gateway



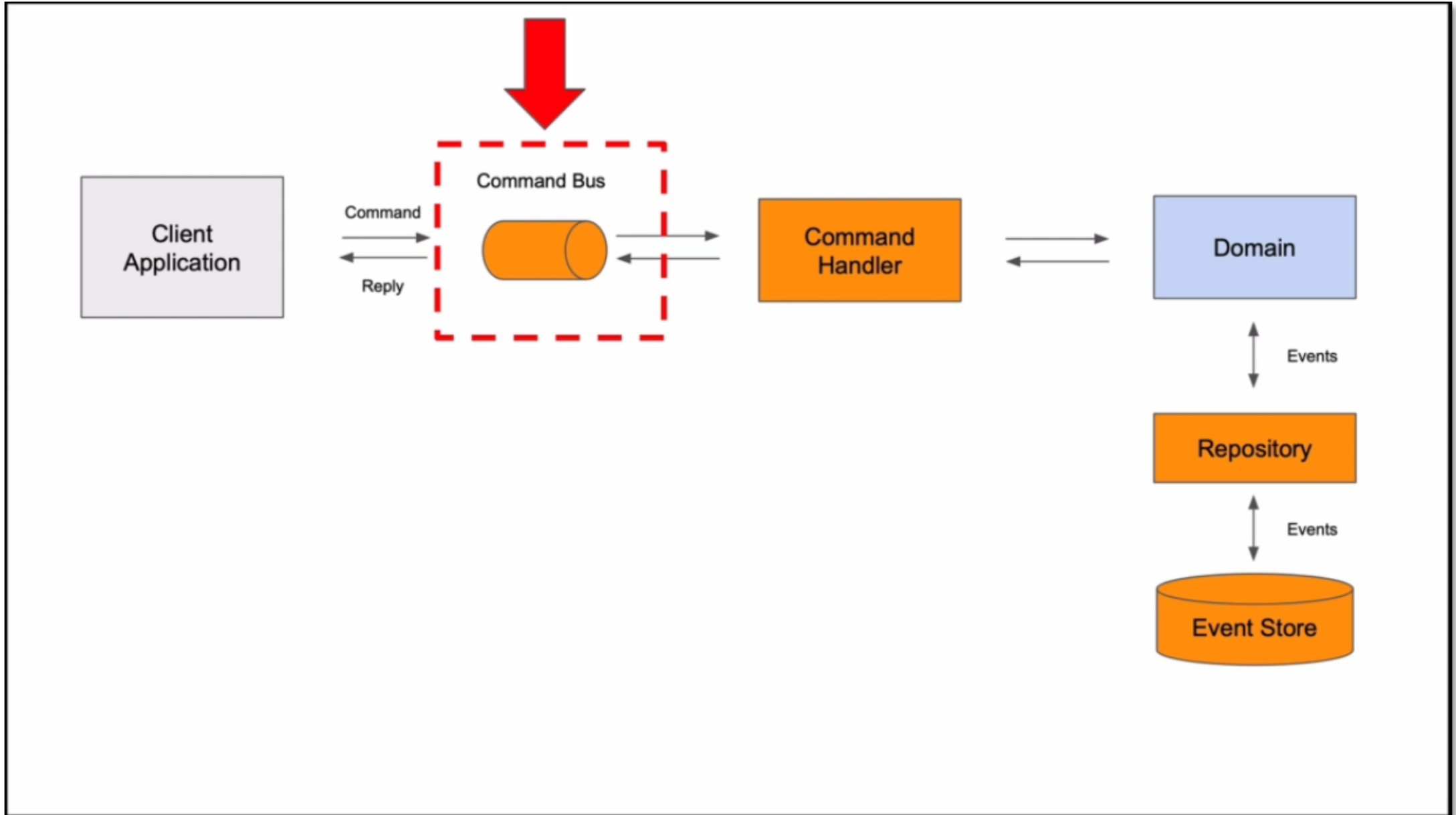
CommandGateway:



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



Send Command to a Command Gateway





Send Command to a Command Gateway

```
ProductController.java
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

ProductCreatedEvent

ProductShippedEvent

ProductDeletedEvent



Creating a new Event class

- Our aggregate will handle the commands, as it's in charge of deciding if a Product can be created, deleted, or shipped.
- It will notify the rest of the application of its decision by publishing an event. We'll have three types of events — ProductCreatedEvent, ProductDeletedEvent, and ProductShippedEvent.



Creating a new Event class

- Let's create the ProductCreatedEvent class:

```
ProductCreatedEvent.java
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
```

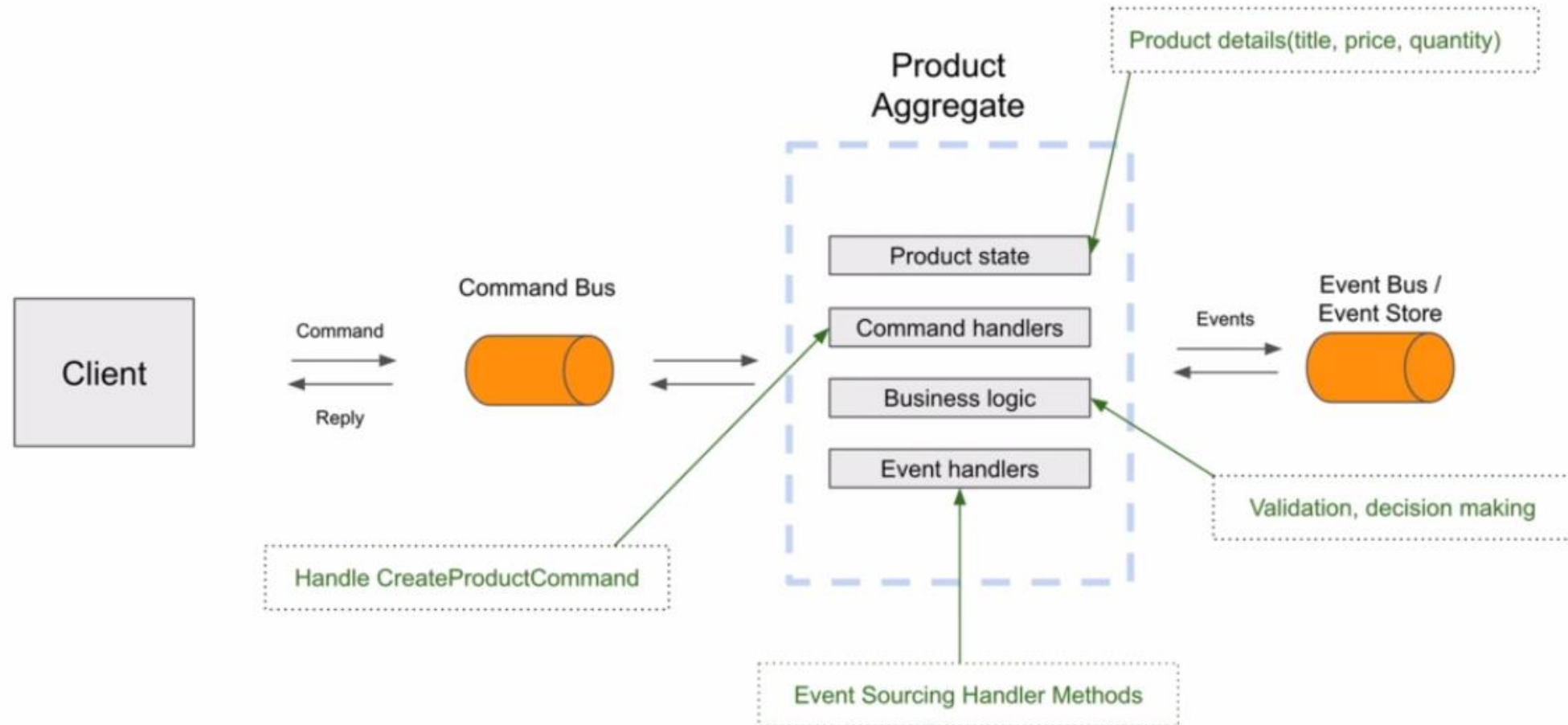


Day - 2

The Command Model – Product Aggregate



Product Aggregate - Introduction





Product Aggregate - Introduction

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



Aggregate



Aggregate



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



CommandHandler



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



Create ProductAggregate class

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

```
ProductAggregate.java
1 package com.mphasis.command;
2
3 import org.axonframework.commandhandling.CommandHandler;
4
5 import org.axonframework.spring.stereotype.Aggregate;
6
7 @Aggregate
8 public class ProductAggregate {
9
10     public ProductAggregate() {
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
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 }
```



@EventSourcingHandler

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



AggregateIdentifier



AggregateIdentifier



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



@EventSourcingHandler

```
ProductAggregate.java
1 package com.mphasis.command;
2
3 import java.math.BigDecimal;
4
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 ProductAggregate() {
24
25     }
26
27     @CommandHandler
28     public ProductAggregate(CreateProductCommand createProductCommand) {
29         // Validate Create Product Command
30
31     }
```



@EventSourcingHandler

```
ProductAggregate.java
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's a navigation bar with 'Home', 'Workspaces', and 'Explore'. A search bar labeled 'Search Postman' is on the right. Below the navigation bar, the main workspace shows a POST request to 'http://localhost:8082/products-service/products'. The request body is a JSON object:

```
{  "title": "iPhone 3",  "price": 300,  "quantity": 2}
```

. The response is displayed at the bottom, showing a status of '200 OK', a time of '1 m 18.78 s', and a size of '152 B'. The response body is a UUID: '51a8bedc-b3ca-436c-b44e-64b70d0750d3'.



Previewing Event in the EventStore

- Go to Axon Server:

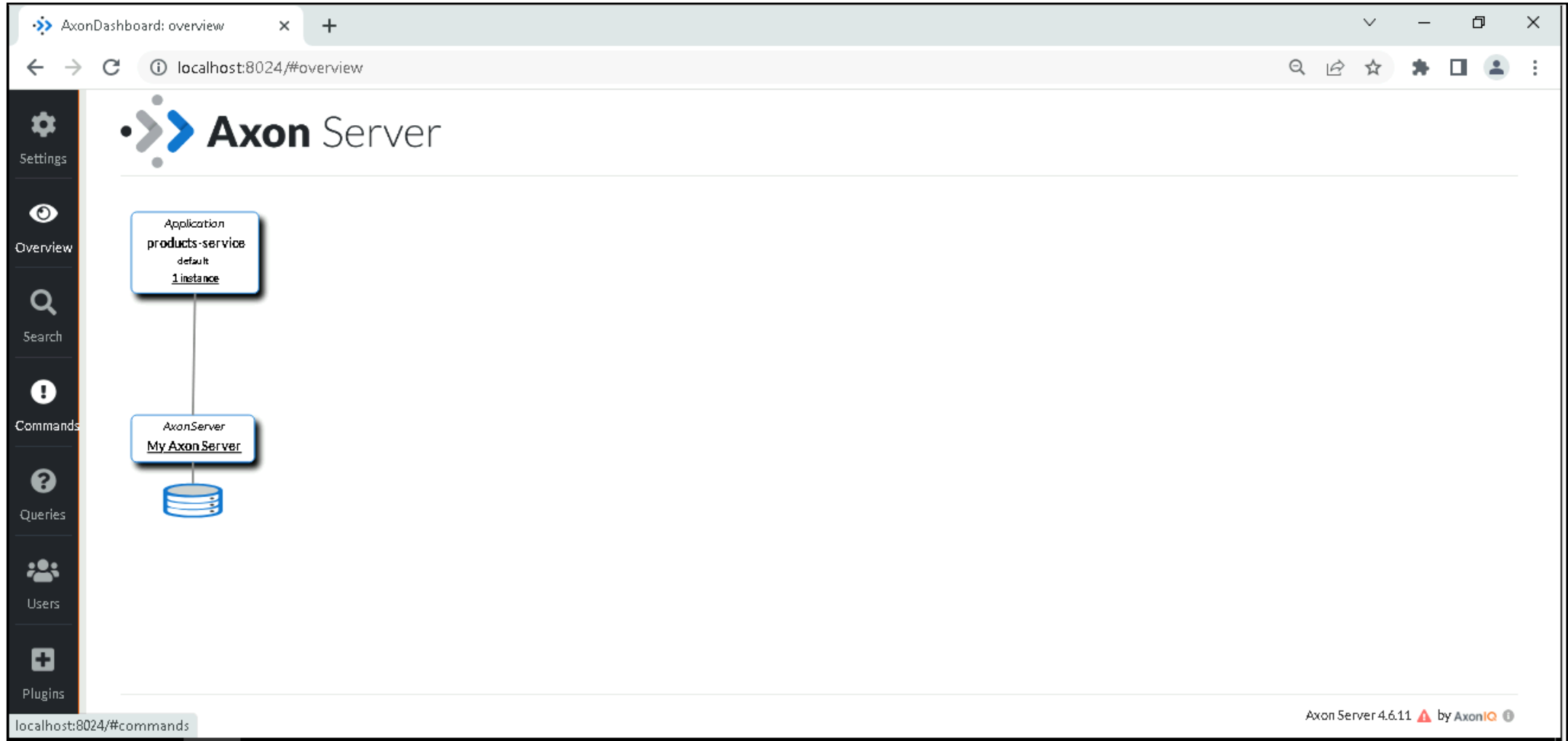
The screenshot shows the Axon Server dashboard in a web browser. The browser's address bar shows 'localhost:8024/#'. The dashboard has a dark sidebar on the left with icons for Settings, Overview, Search, Commands, Queries, Users, and Plugins. The main content area is titled 'Axon Server' and contains three yellow warning boxes: 'SSL disabled', 'Development Mode is enabled', and 'Authentication disabled'. Below these are three panels: 'Configuration', 'Status', and 'License'. The 'Configuration' panel lists Node Name (My Axon Server), Host Name (localhost), Http Port (8024), and GRPC Port (8124). The 'Status' panel shows 'Last event token' as 0 and 'Activity in the last minute' with metrics for Commands received / second, Queries received / second, Events stored / second, and Snapshots stored / second, all showing 0. There is a 'Reset Event Store' button at the bottom of the Status panel. The 'License' panel shows 'Edition' as 'Standard Edition'.

Configuration		Status		License	
Node Name	My Axon Server	Last event token	0	Edition	Standard Edition
Host Name	localhost	Activity in the last minute			
Http Port	8024	Commands received / second	0		
GRPC Port	8124	Queries received / second	0		
		Events stored / second	0		
		Snapshots stored / second	0		



Previewing Event in the EventStore

- Click on **Overview** tab:





Previewing Event in the EventStore

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

The screenshot shows the Axon Server dashboard in a web browser. The browser tab is labeled 'AxonDashboard: overview' and the address bar shows 'localhost:8024/#overview'. The dashboard has a dark sidebar with navigation icons for Settings, Overview, Search, Commands, Queries, Users, and Plugins. The main content area displays 'Axon Server' at the top, followed by 'Application details for products-service'. Below this is a section for 'Subscription Queries' with three metrics: #Total Subscription Queries (0), #Active Subscription Queries (0), and #Updates to Subscription Queries (0). There are two tables below. The first table has columns 'Instance Name', 'AxonServer Node', and 'Tags', with one row showing '3656@microservices', 'My Axon Server ← default', and an empty tag field. The second table has columns 'Command' and 'Tags', with one row showing 'com.mphasis.command.CreateProductCommand' and an empty tag field. At the bottom, there is a table with columns 'Query', 'Response Types', '#Subscriptions', '#Active Subscriptions', and '#Updates', which is currently empty. The footer of the dashboard indicates 'Axon Server 4.6.11 by AxoniQ'.

Instance Name	AxonServer Node	Tags
3656@microservices	My Axon Server ← default	

Command	Tags
com.mphasis.command.CreateProductCommand	

Query	Response Types	#Subscriptions	#Active Subscriptions	#Updates
-------	----------------	----------------	-----------------------	----------



Previewing Event in the EventStore

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

The screenshot shows the Axon Server Search interface in a web browser. The browser tab is labeled 'AxonDashboard: query' and the address bar shows 'localhost:8024/#query'. The interface has a dark sidebar on the left with navigation links: Settings, Overview, Search (active), Commands, Queries, Users, and Plugins. The main content area displays the 'Axon Server' logo and search options. The 'Search' tab is selected, showing a search bar with the placeholder 'Enter your query here' and an orange 'Search' button. Above the search bar, there are radio buttons for 'Events' (selected) and 'Snapshots', and a 'Query time window' dropdown set to 'last hour' with a 'Live Updates' checkbox. Below the search bar, there is a link 'About the query language'. The search results are displayed in a table with the following columns: token, eventIdIdentifier, aggregateIdenti..., aggrega..., aggregateType, payloadType, payload..., payloadData, timestamp, and metaData. The first row shows a token of '0', an eventIdIdentifier of 'f6ce3e9c-3fe...', an aggregateIdenti... of '51a8bedc-b3...', an aggrega... of '0', an aggregateType of 'ProductAggr...', a payloadType of 'com.mphasis.events.Product...', a payload... of '<com.mphasis.events.ProductCreatedEve...', a timestamp of '2023-07-04...', and a metaData of '{traceld=c...'. At the bottom right of the table, it says 'Rows per page 10 1 - 1 of 1'. The footer of the page indicates 'Axon Server 4.6.11 by AxoniQ'.

token	eventIdIdentifier	aggregateIdenti...	aggrega...	aggregateType	payloadType	payload...	payloadData	timestamp	metaData
0	f6ce3e9c-3fe...	51a8bedc-b3...	0	ProductAggr...	com.mphasis.events.Product...	<com.mphasis.events.ProductCreatedEve...		2023-07-04...	{traceld=c...



Previewing Event in the EventStore

- About the **query** language:

AxonDashboard: query

localhost:8024/#query

Enter your query here

Search

About the query language

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

Filtering

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"
```

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 Server web interface in a browser window. The address bar shows 'localhost:8024/#query'. The interface has a sidebar on the left with navigation links: Settings, Overview, Search, Commands, Queries, Users, and Plugins. The main content area is titled 'Axon Server' and has a search bar with the query 'aggregateIdentifier = "51a8bedc-b3ca-436c-b44e-64b70d0750d3"'. The search results show a table with columns: eventIdentifier, aggregateIdentifier, aggregateType, payloadType, payloadR..., payloadData, timestamp, and metaData. The first row shows an event with identifier 'f6ce3e9c-3fed...' and aggregate identifier '51a8bedc-b3ca-436c-b44e-64b70d0750d3'. The event type is 'ProductAggregate' and the payload type is 'com.mphasis.events.ProductCr...'. The timestamp is '2023-07-04T...' and the metaData is '{traceId=c...'. The bottom of the interface shows 'Axon Server 4.6.11 by AxonIQ'.

AxonDashboard: query x +

localhost:8024/#query

Axon Server

Search: ☒ Events ☐ Snapshots Query time window: custom ☒ Live Updates

aggregateIdentifier = "51a8bedc-b3ca-436c-b44e-64b70d0750d3" Search

About the query language

eventIdentifier	aggregateIdentifier	aggregateType	payloadType	payloadR...	payloadData	timestamp	metaData
f6ce3e9c-3fed...	51a8bedc-b3ca-436c-b44e-64b70d0750d3	0	ProductAggregate	com.mphasis.events.ProductCr...	<com.mphasis.events.ProductCreatedEvent>...	2023-07-04T...	{traceId=c...

Rows per page 10 1 - 1 of 1

localhost:8024/#query 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 Event Store interface. The left sidebar contains navigation links: Settings, Overview, Search, Commands, Queries, Users, and Plugins. The main area displays a search for 'Events' with a query window set to 'custom' and 'Live Updates' checked. A search button is visible. Below the search bar, a table lists event details. A modal window is open, showing the following event details:

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

The bottom of the interface shows 'Axon Server 4.6.11 by AxonIQ'.



Previewing Event in the EventStore

- Go to Commands tab:

The screenshot shows the Axon Server dashboard in a web browser. The browser tab is labeled 'AxonDashboard: commands' and the address bar shows 'localhost:8024/#commands'. The dashboard has a dark sidebar with navigation icons for Settings, Overview, Search, Commands (selected), Queries, Users, and Plugins. The main content area displays the 'Axon Server' logo and a table of commands. The table has two columns: 'Commands' and a count. One command is listed: 'com.mphasis.command.CreateProductCommand' with a count of '1'. The command is associated with the user 'products-service@default' and '3656@microservices'. The footer of the dashboard indicates 'Axon Server 4.6.11 by AxonIQ'.

Commands	
com.mphasis.command.CreateProductCommand	1



Recap of Day – 2

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



Recap of Day – 2

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



THANK YOU

About Mphasis

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

Important Confidentiality Notice

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

