

16장. API Composition



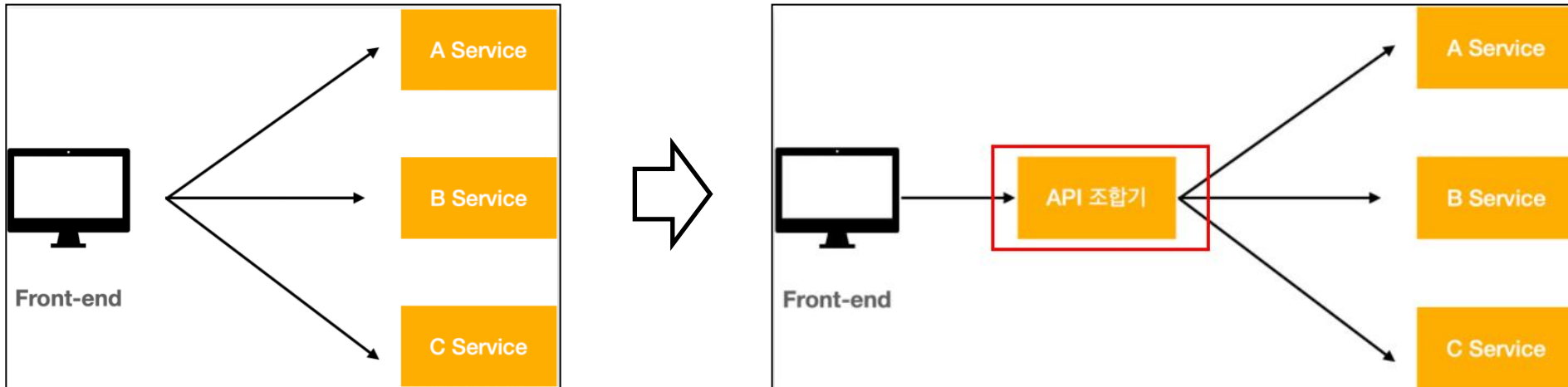
API Composition 개요/특징

API Composition 구축 방법

1. API Composition 개요

1. 개요

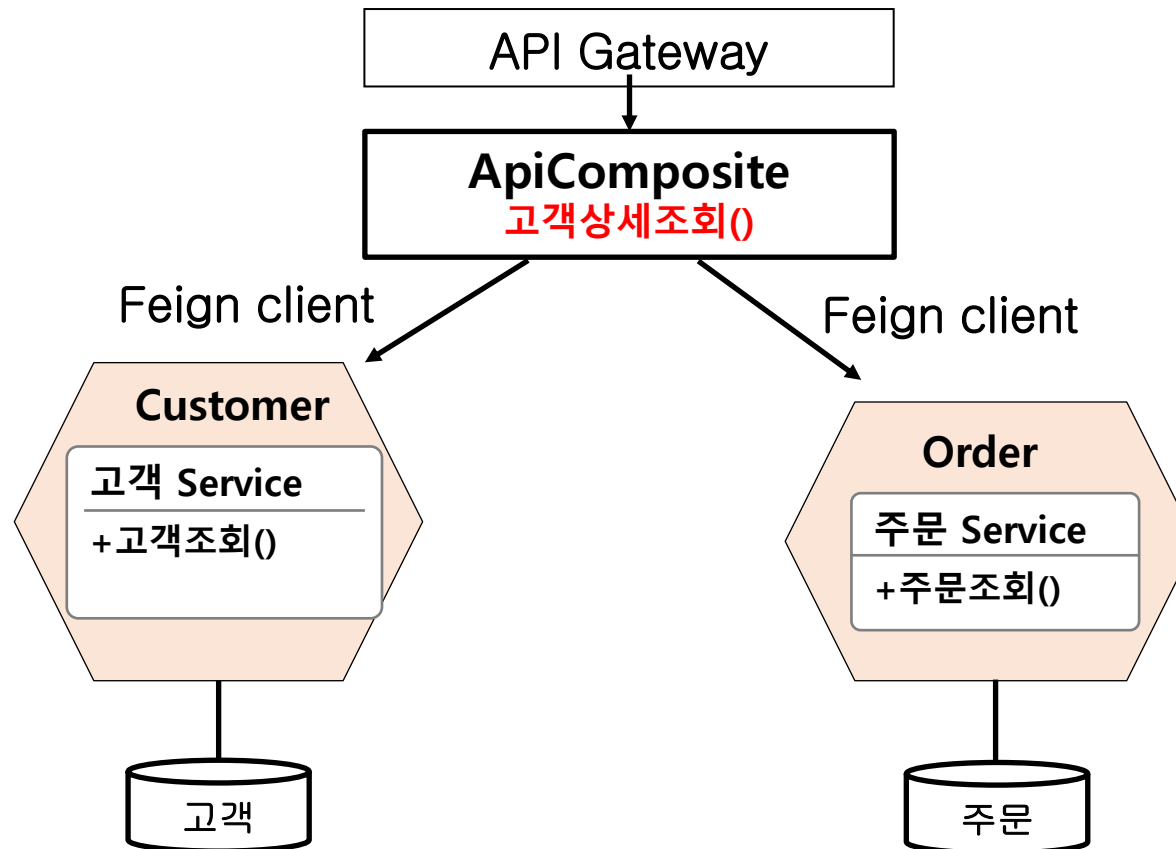
하나의 DB에 여러 데이터를 저장하던 모놀리식 아키텍처에서는 여러 도메인의 데이터들을 조회하고 DB에서 쿼리로 조인하여 처리하기는 쉽다.
하지만 MSA 환경에서는 여러 도메인의 조합된 데이터를 조회하기는 쉽지 않다.



2. API Composition 구축 실습 개요 – server side 패턴

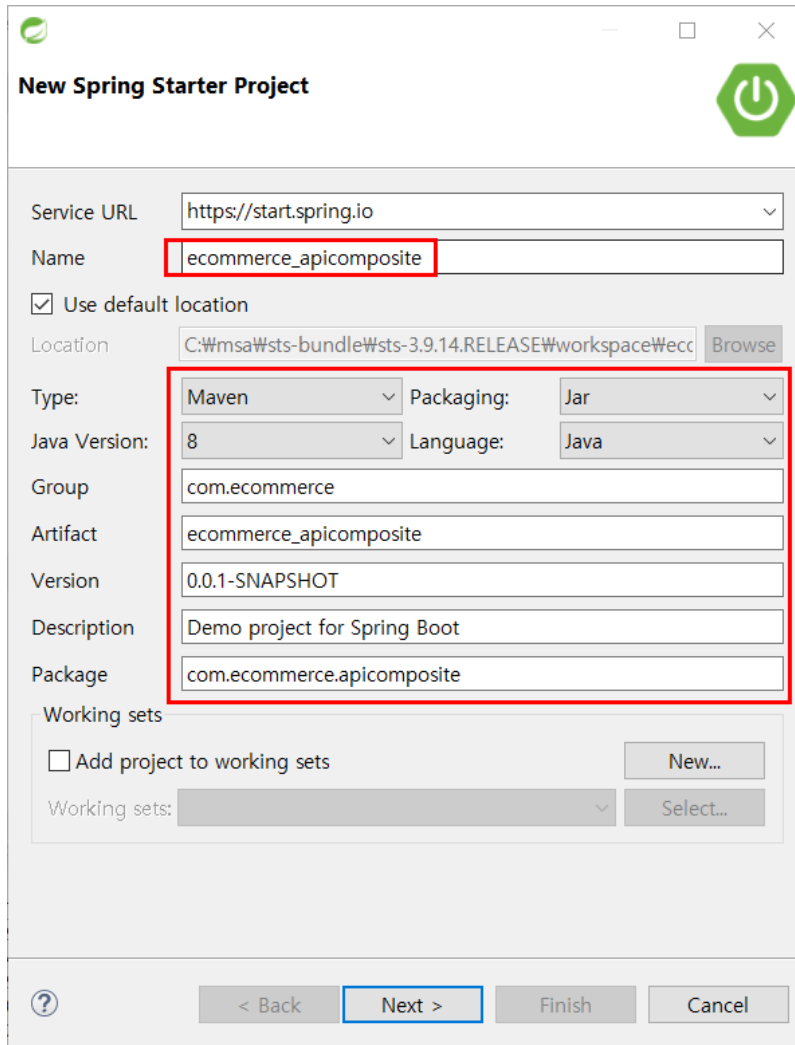
1) 개요

고객정보 조회시 기본 고객정보 외에 고객의 주문정보를 함께 조회하고자 한다. 이를 위해 Customer의 고객상세조회() 에서 Order의 주문조회() API를 호출하고 그 결과를 Composite하여 Server Side 패턴을 적용하여 제공하도록 구현한다.



2. API Composition 구축 실습 개요 – server side 패턴

1) ecommerce_apicomposite 프로젝트 작성



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

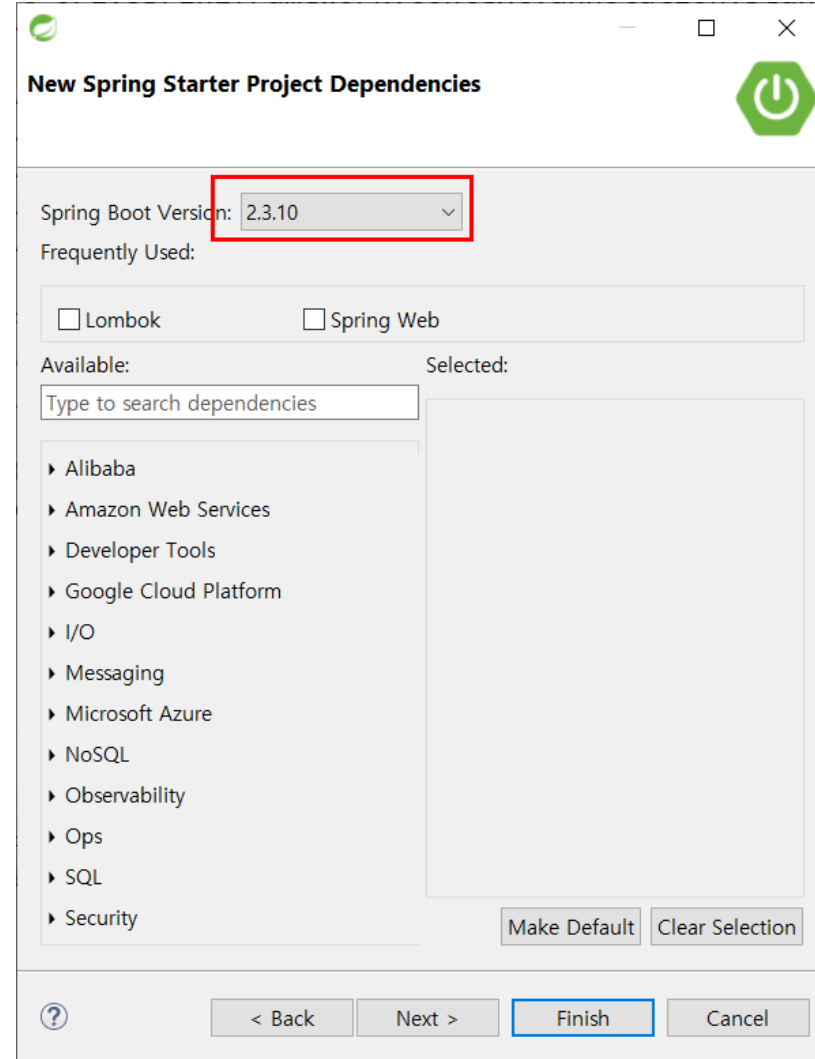
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



New Spring Starter Project Dependencies

Spring Boot Version:

Frequently Used:

☐ Lombok ☐ Spring Web

Available:

Selected:

- ▶ Alibaba
- ▶ Amazon Web Services
- ▶ Developer Tools
- ▶ Google Cloud Platform
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security

2. API Composition 구축 실습 개요 – server side 패턴

2) pom.xml 파일에 dependency 설정

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
  <version>2.2.8.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
  <version>2.2.8.RELEASE</version>
</dependency>
```

2. API Composition 구축 실습 개요 – server side 패턴

2) CustomerFeignClient 설정

```
@FeignClient(name="ecommerce-customer",
            fallbackFactory = CustomerFeignClientFallbackFactory.class
        )
public interface CustomerFeignClient {
    @GetMapping("/ecommerce/customer/rest/customers/{userid}")
    public Customer selectCustomerByUserId(@PathVariable String userid)throws Exception;
}

@Component
class CustomerFeignClientFallbackFactory implements FallbackFactory<CustomerFeignClient>{
    @Override
    public CustomerFeignClient create(Throwable cause) {
        return new CustomerFeignClient() {
            private final Logger LOGGER = LoggerFactory.getLogger(CustomerFeignClient.class);
            @Override
            public Customer selectCustomerByUserId(String userid) throws Exception {
                String msg = "feignClient를 이용한 Customer 서비스 호출에 문제가 있습니다.";
                LOGGER.info(msg, cause);
                throw new Exception(msg);
            }
        };
    }
}
```

2. API Composition 구축 실습 개요 – server side 패턴

3) OrderFeignClient 설정

```
@FeignClient(name="ecommerce-order",
            fallbackFactory = OrderFeignClientFallbackFactory.class
        )
public interface OrderFeignClient {
    @GetMapping("/ecommerce/order/rest/orders/{userid}")
    public List<Order> retrieveCustomer(@PathVariable String userid)throws Exception;
}

@Component
class OrderFeignClientFallbackFactory implements FallbackFactory<OrderFeignClient>{
    @Override
    public OrderFeignClient create(Throwable cause) {
        return new OrderFeignClient() {
            private final Logger LOGGER = LoggerFactory.getLogger(OrderFeignClient.class);
            @Override
            public List<Order> retrieveCustomer(String userid) throws Exception {
                String msg = "feignClient를 이용한 Order 서비스 호출에 문제가 있습니다.";
                LOGGER.info(msg, cause);
                throw new Exception(msg);
            }
        };
    }
}
```

2. API Composition 구축 실습 개요 – server side 패턴

4) ApiCompositeServiceImpl 서비스 구현

```
@Service("apiCompositeService")
public class ApiCompositeServiceImpl implements ApiCompositeService {

    @Autowired
    OrderFeignClient orderFeignClient;

    @Autowired
    CustomerFeignClient customerFeignClient;

    @Override
    public Customer retrieveCustomerDetail(String userid) throws Exception {

        //1. 고객 기본 정보
        Customer customer = customerFeignClient.selectCustomerByUserId(userid);
        //2. 고객 주문 정보
        List<Order> orderList = orderFeignClient.retrieveCustomer(userid);
        customer.setOrderList(orderList);

        return customer;
    }
}
```


2. API Composition 구축 실습 개요 – server side 패턴

5) ApiCompositeController 작성

```
@RestController
public class ApiCompositeController {

    @Autowired
    ApiCompositeService apiCompositeService;

    @ApiOperation(value = "고객상세조회", httpMethod = "GET", notes = "고객상세조회")
    @RequestMapping(method = RequestMethod.GET, path = "/detail/rest/customers/{userid}")
    public Customer retrieveCustomerDetail(@PathVariable(name = "userid") String userid) throws Exception{
        return apiCompositeService.retrieveCustomerDetail(userid);
    }
}
```

6) application.properties

```
server.port=8089
server.servlet.context-path=/ecommerce/apicomposite
#spring
spring.application.name=ecommerce-apicomposite

# eureka
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
eureka.client.service-url.defaultZone=http://${EUREKA}:8761/eureka
eureka.instance.prefer-ip-address=true

# Env
POSTGRES=localhost
EUREKA=localhost
```

7) Application 수정

```
@SpringBootApplication
@EnableFeignClients
public class EcommerceApicompositeApplication {

    public static void main(String[] args) {
        SpringApplication.run(EcommerceApicompositi
    }
}
```

2. API Composition 구축 실습 개요 – server side 패턴

8) 실행 및 Swagger UI 요청

<http://localhost:8089/ecommerce/apicomposite/swagger-ui.html>

api-composite-controller Api Composite Controller

GET /detail/rest/customers/{userid} 고객상세조회

고객상세조회

Parameters

Name	Description
userid * required	userid
string (path)	<input type="text" value="1111"/>

Code	Details
200	<div>Response body<pre>{ "userid": "1111", "pwd": "1234", "name": "홍길동", "email": "hong@korea.com", "createDate": "2021-05-16", "orderList": [{ "orderId": "A1111", "userId": "1111", "name": "홍길동", "productId": "CATALOG-0001", "quantity": 10, "unitPrice": 500, "totalPrice": 5000, "create_date": "2021-05-16" }, { "orderId": "B1111", "userId": "1111", "name": "홍길동", "productId": "CATALOG-0002", "quantity": 5, "unitPrice": 200, "totalPrice": 1000, "create_date": "2021-05-16" }] }</pre></div>

17장. CQRS



CQRS 개요/적용

CQRS 아키텍처

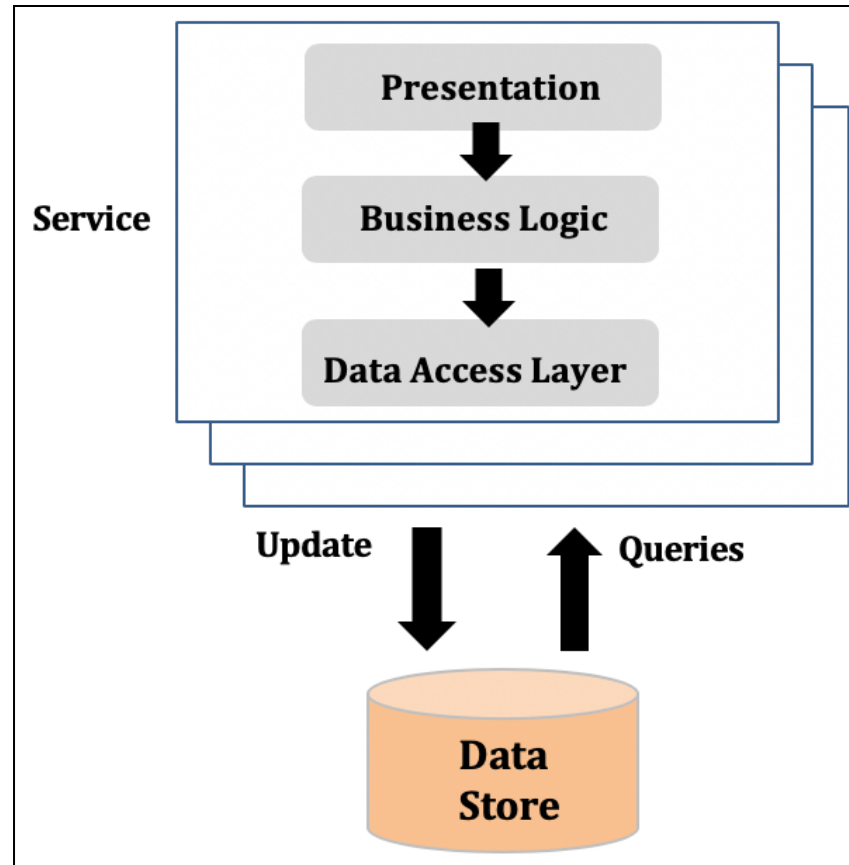
API Composition & CQRS

1. CQRS 개요

1. 개요 및 문제점

CQRS (Command Query Responsibility Segregation)는 읽기와 쓰기를 분리하는 패턴을 의미한다.

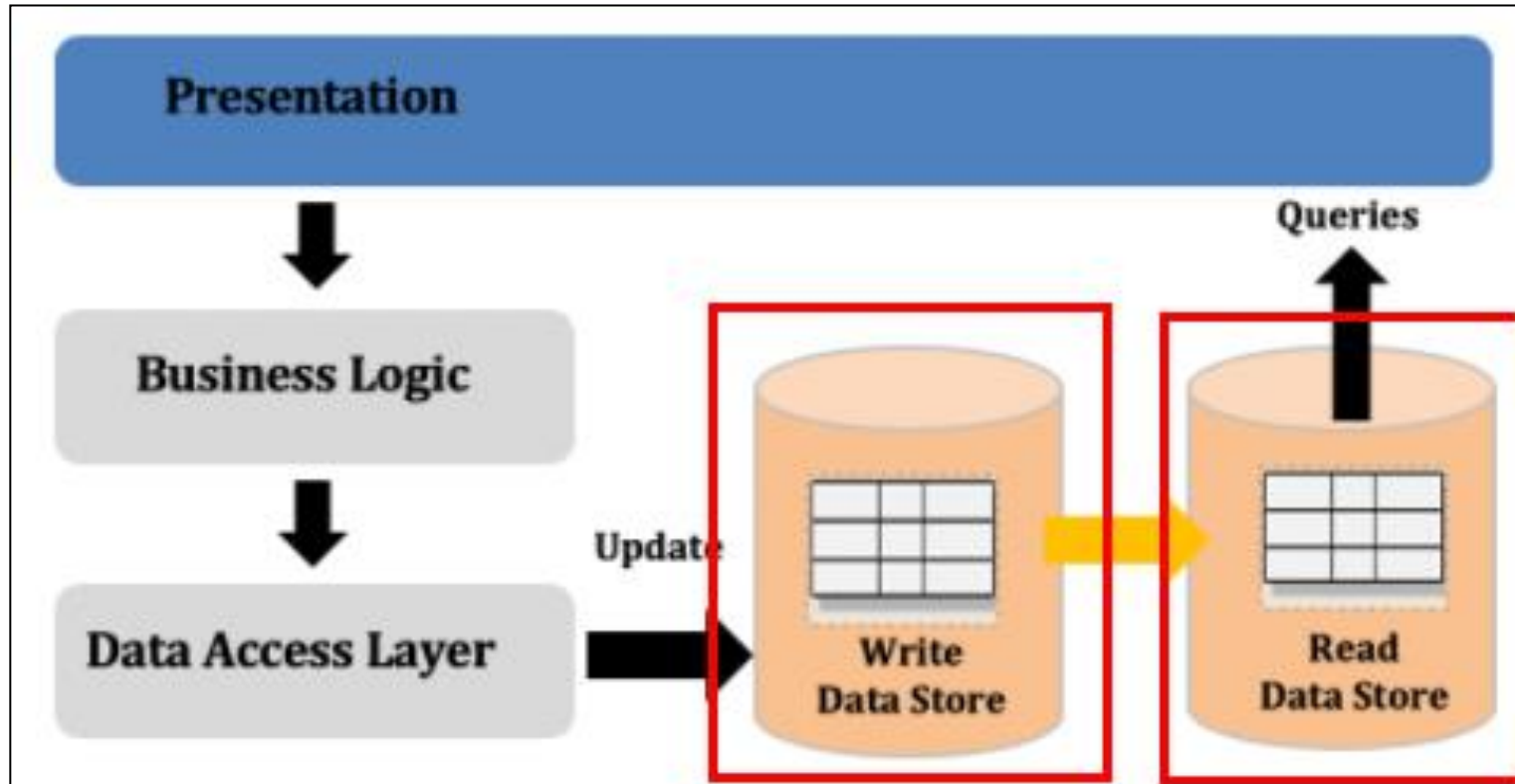
다음 그림은 마이크로서비스의 핵심철학으로 서비스별 데이터 저장소 패러다임을 구축한 것이다. 그런데 서비스의 성능향상을 위해서 서비스 인스턴스 (instance)를 Scale-out하여 여러 개로 실행한 경우에 데이터 읽기/쓰기 작업으로 인해서 리소스 교착 상태가 발생할 수 있다. 이러한 문제를 해결하기 위한 방법이 CQRS 패턴이다.



2. CQRS 적용

2. CQRS 적용

다음 그림과 같이 쓰기 트랜잭션용 저장소와 조회용 저장소를 따로 분리하여 구축한다. 이렇게 쓰기와 조회의 전략을 각각 분리하면, 쓰기 시스템의 부하를 줄이고 조회 대기 시간을 줄이는 등 많은 장점을 가질 수 있다.



3. CQRS 아키텍처

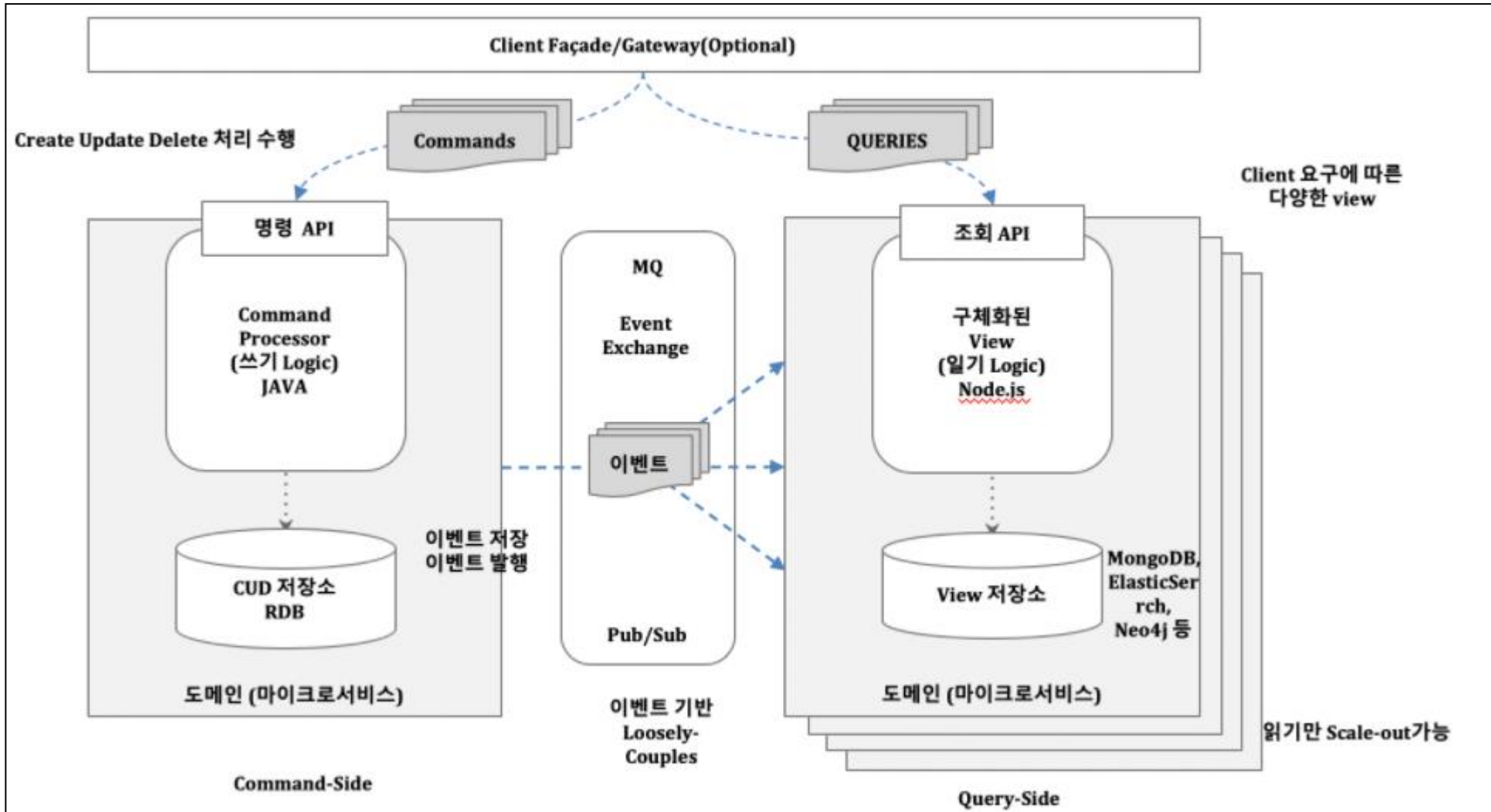
3. CQRS 아키텍처

명령 측면의 마이크로서비스는 입력, 수정, 삭제처리를 수행하고 저장소로는 쓰기에 최적화된 RDB를 사용한다.

반면에 쿼리 측면의 마이크로서비스는 조회성능이 좋은 MongoDB나 Elasticsearch같은 NoSQL를 사용할 수 있는 형태로 구축할 수 있다.

주요한 문제는 명령 측면의 서비스가 사용됨에 따라서 조회 측면 서비스와의 일관성이 깨지게 된다. 따라서 데이터 일관성 유지를 위해서 Pub/Sub 아키텍처를 사용한다.

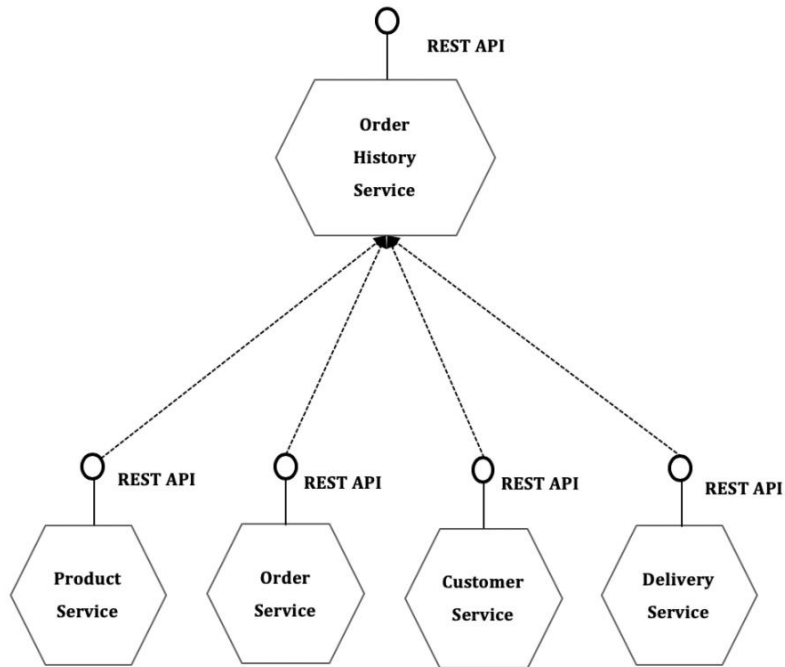
3. CQRS 아키텍처



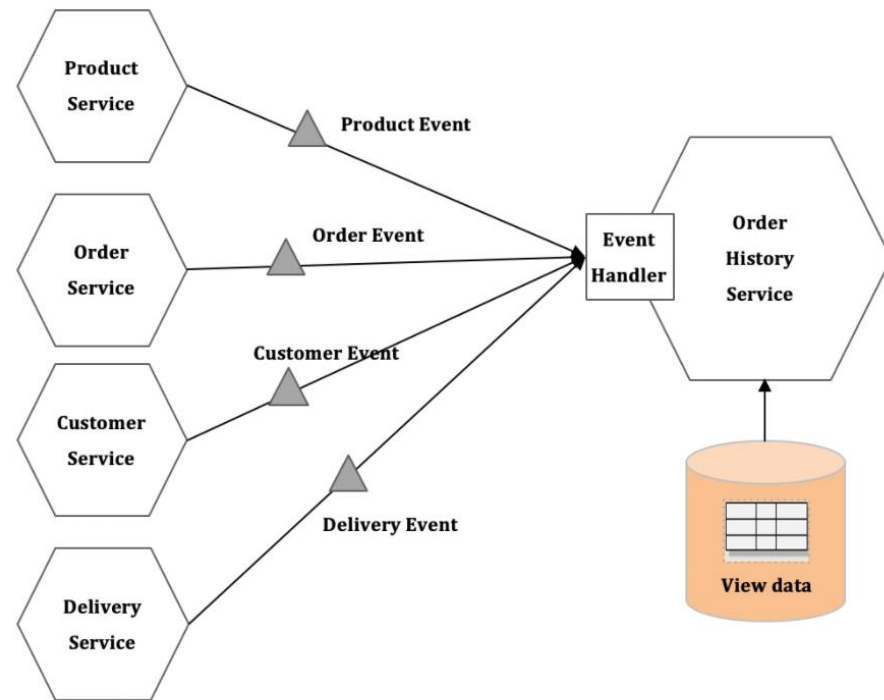
4. API Composition과 CQRS

여러 개의 마이크로서비스들이 연계되어 서비스를 제공해야 되는 경우의 해결책은 다음과 같다.

가) API Composition



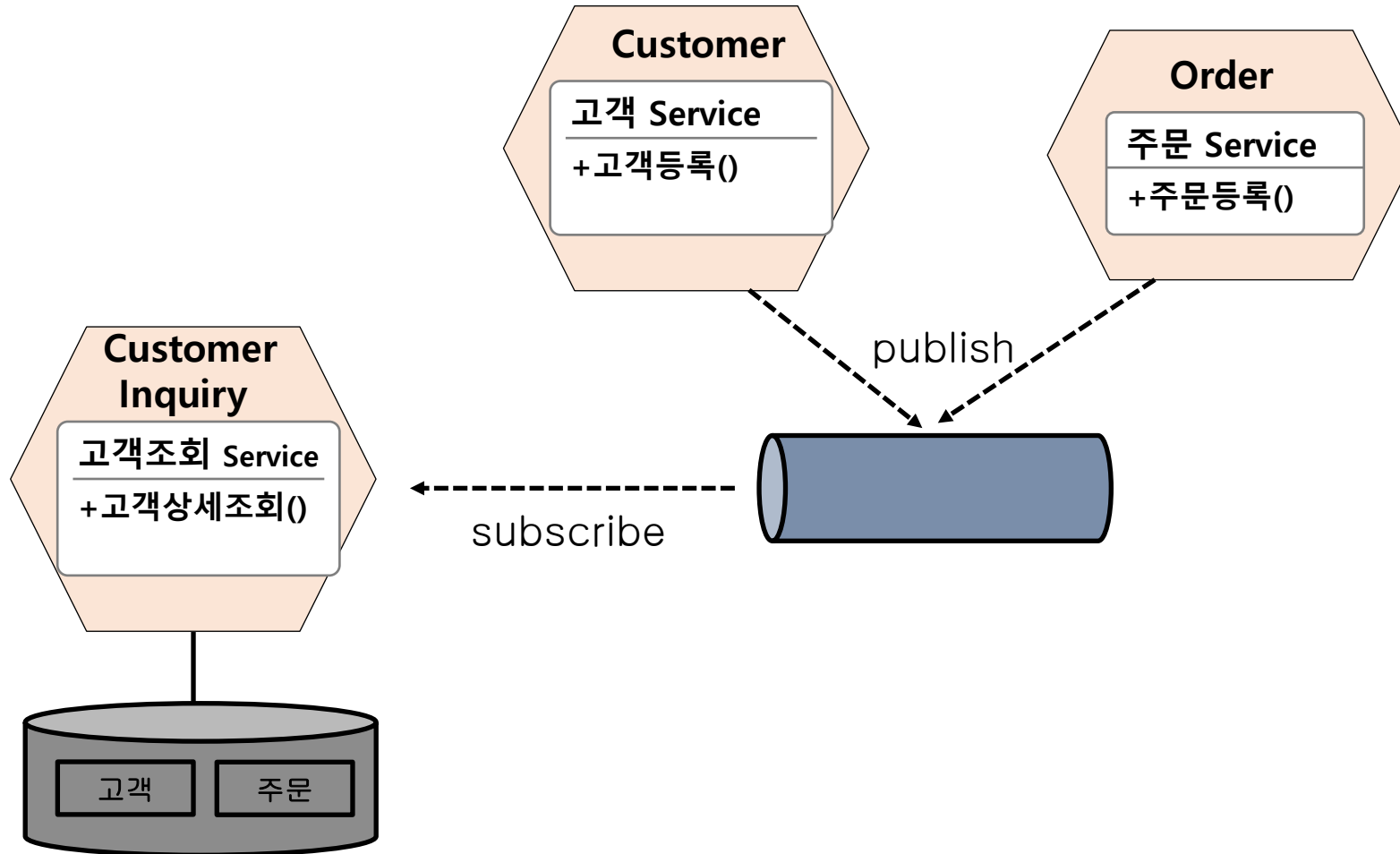
나) CQRS



5. CQRS 구축 실습

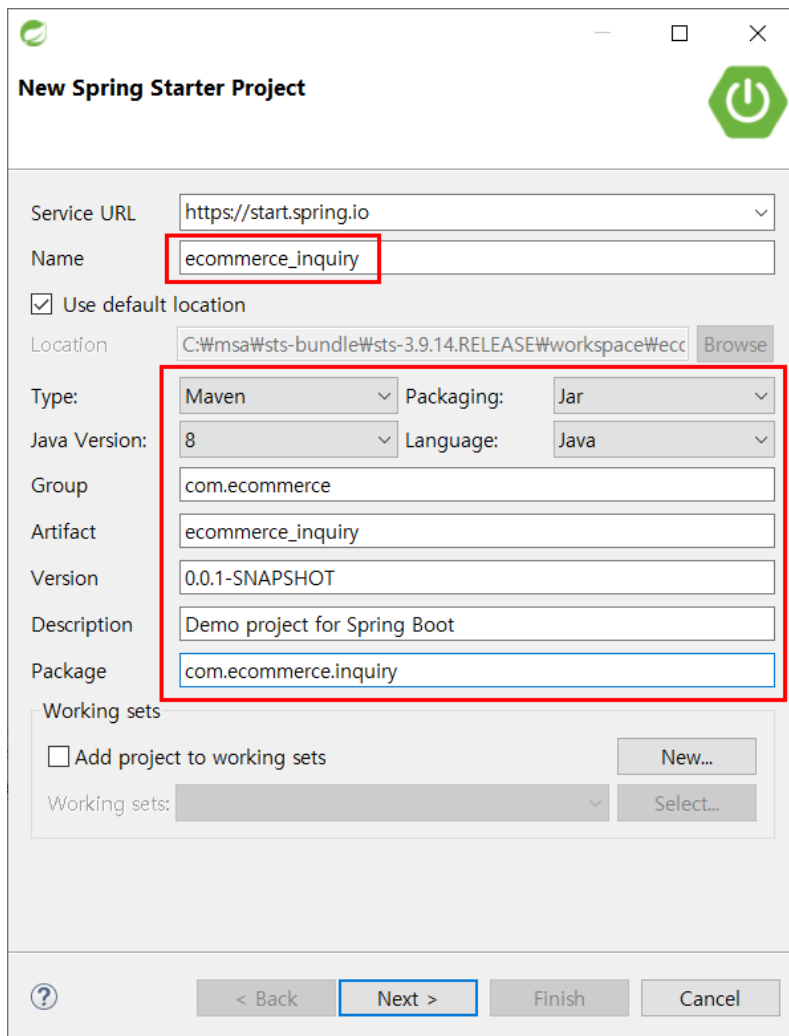
1) 개요

앞에서 API Composition 방식으로 개발한 고객상세조회()를 Customer Inquiry 서비스에서 CQRS 방식으로 구현하고 테스트한다.



5. CQRS 구축 실습

1) ecommerce_inquiry 프로젝트 작성



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

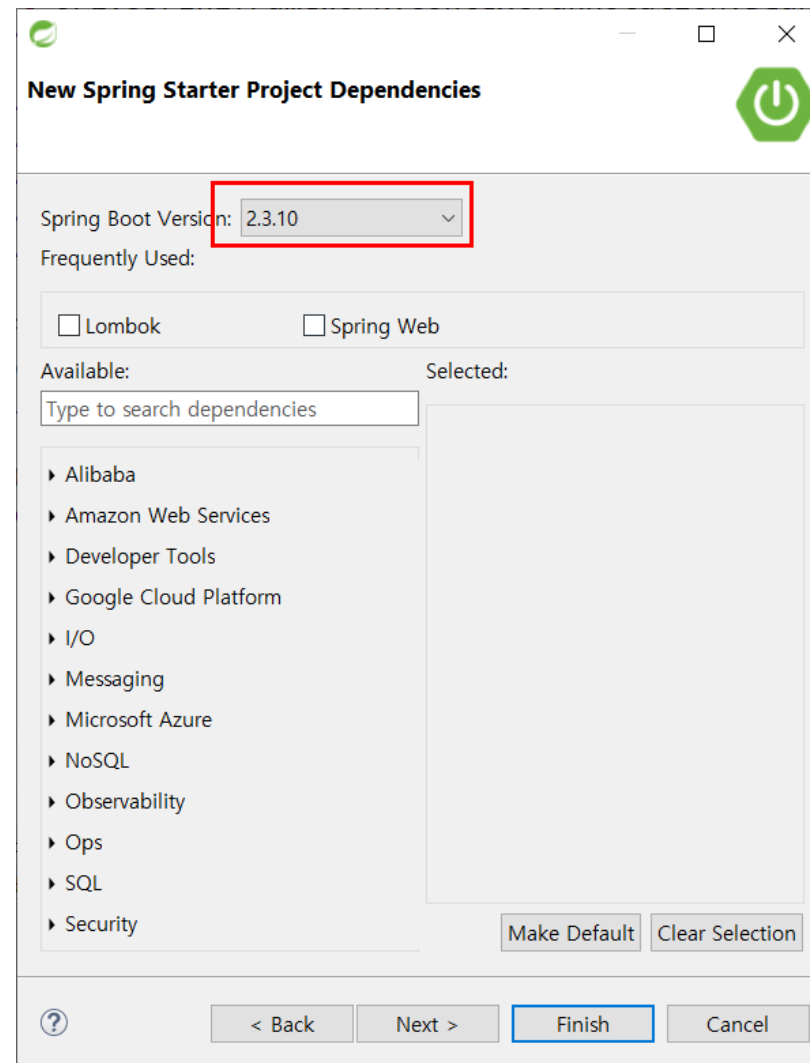
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



New Spring Starter Project Dependencies

Spring Boot Version:

Frequently Used:

☐ Lombok ☐ Spring Web

Available:

Type to search dependencies

- ▶ Alibaba
- ▶ Amazon Web Services
- ▶ Developer Tools
- ▶ Google Cloud Platform
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security

Selected:

5. CQRS 구축 실습

2) pom.xml 파일에 dependency 설정

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>2.1.4</version>
</dependency>
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
</dependency>
<dependency>
  <groupId>org.bgee.log4jdbc-log4j2</groupId>
  <artifactId>log4jdbc-log4j2-jdbc4</artifactId>
  <version>1.16</version>
</dependency>

<dependency>
  <groupId>org.modelmapper</groupId>
  <artifactId>modelmapper</artifactId>
  <version>2.4.1</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```

```
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
```

5. CQRS 구축 실습

3) com.ecommerce.inquiry.domain.entity.**Customer** 빈 생성

```
@Alias("Customer")
public class Customer implements Serializable{

    private String userid;
    private String pwd;
    private String name;
    private String email;
    private String createDate;
```

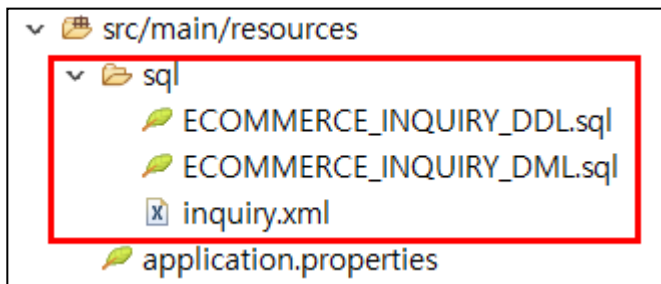
4) com.ecommerce.inquiry.domain.entity.**Order** 빈 생성

```
@Alias("Order")
public class Order implements Serializable{

    private String orderId; //pk
    private String userId;
    private String name;
    private String productId;
    private int quantity;
    private int unitPrice;
    private int totalPrice;
    private String create_date;
```

5. CQRS 구축 실습

5) 제공된 sql 파일과 MyBatis mapper 파일 복사



```
<mapper namespace="com.ecommerce.inquiry.domain.repository.InquiryRepository">
  <insert id="insertOrder" parameterType="Order">
    insert into MSA_ORDER_CQRS ( orderId, userId, name, productId, quantity, unitPrice, totalPrice)
    values (#{orderId}, #{userId},#{name}, #{productId}, #{quantity}, #{unitPrice}, #{totalPrice})
  </insert>
  <insert id="insertCustomer" parameterType="Customer">
    insert into MSA_CUSTOMER_CQRS ( userid, pwd, name, email)
    values (#{userid}, #{pwd}, #{name}, #{email})
  </insert>
  <select id="selectCustomerByUserId" resultType="Customer"
    parameterType="string">
    select userid, pwd, name, email, create_date as createDate
    from MSA_CUSTOMER_CQRS
    where userid = #{userid}
  </select>
  <select id="selectOrderByUserId" resultType="Order" parameterType="string">
    select orderId, userId, name, productId, quantity, unitPrice, totalPrice, create_date
    from MSA_ORDER_CQRS
    where userId = #{userId}
  </select>
</mapper>
```

5. CQRS 구축 실습

6) com.ecommerce.inquiry.service.InquiryService 인터페이스 작성

```
public interface InquiryService {  
  
    public Customer retrieveCustomerDetail(String userid) throws Exception;  
    public Customer selectCustomerByUserId(String userid) throws Exception;  
    public int insertCustomer(Customer customer) throws Exception;  
    public List<Order> selectOrderByUserId(String userid) throws Exception;  
    public int insertOrder(Order order) throws Exception;  
}
```

7) com.ecommerce.inquiry.service.InquiryServiceImpl 작성

```
@Service("inquiryService")  
public class InquiryServiceImpl implements InquiryService {  
  
    @Autowired  
    InquiryRepository inquiryRepository;  
  
    @Override  
    public Customer retrieveCustomerDetail(String userid) throws Exception {  
  
        // 1. 고객 기본 정보  
        Customer customer = inquiryRepository.selectCustomerByUserId(userid);  
        // 2. 고객 주문 정보  
        List<Order> orderList = inquiryRepository.selectOrderByUserId(userid);  
        customer.setOrderList(orderList);  
  
        return customer;  
    }  
}
```

5. CQRS 구축 실습

8) com.ecommerce.inquiry.controller.InquiryController 빈 생성

```
@RestController
public class InquiryController {

    @Autowired
    private InquiryService inquiryService;

    @ApiOperation(value = "고객상세조회", httpMethod = "GET", notes = "고객상세조회")
    @RequestMapping(method = RequestMethod.GET, path = "/detail/rest/customers/{userid}")
    public Customer retrieveCustomerDetail(@PathVariable(name = "userid") String userid)
        return inquiryService.retrieveCustomerDetail(userid);
    }
}
```

5. CQRS 구축 실습

9) application.properties 파일에 속성값 설정

```
application.properties
1 server.port=8078
2 server.servlet.context-path=/ecommerce/inquiry
3 #spring
4 spring.application.name=ecommerce-inquiry
5
6 # PostgreSQL
7 spring.datasource.driver-class-name=net.sf.log4jdbc.sql.jdbcapi.DriverS
8 spring.datasource.url=jdbc:log4jdbc:postgresql://${POSTGRES}:5432/cqrs
9 spring.datasource.username=postgres
10 spring.datasource.password=admin1234
11 spring.datasource.schema=classpath:sql/ECOMMERCE_INQUIRY_DDL.sql
12 spring.datasource.data=classpath:sql/ECOMMERCE_INQUIRY_DML.sql
13 spring.datasource.initialization-mode=always
14
15 # MyBatis
16 mybatis.type-aliases-package=com.ecommerce.inquiry.domain.entity
17 mybatis.mapper-locations=classpath:sql/*.xml
18
19 # eureka
20 eureka.client.register-with-eureka=true
21 eureka.client.fetch-registry=true
22 eureka.client.service-url.defaultZone=http://${EUREKA}:8761/eureka
23 eureka.instance.prefer-ip-address=true
24
25 # Env
26 POSTGRES=localhost
27 EUREKA=localhost
28
29 # actuator \uC5D4\uB4DC\uD3EC\uC778\uD2B8 \uB178\uC9C
30 # https://docs.spring.io/spring-boot/docs/2.3.10.RELEASE/reference/html
31 management.endpoints.web.exposure.include=info,health,beans,metrics
```


5. CQRS 구축 실습

10) KafkaSubscriberConfig

```
@Configuration
public class KafkaSubscriberConfig {

    public ConsumerFactory<String, Customer> customerConsumerFactory() {
        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "customer_consumer"); // Consumer를 식별하는 고유 아이디.
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false"); // offset을 주기적으로 commit 할지 여부
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest"); // earliest: 맨 처음부터 다시, latest: 이전꺼 무시 새로입력 데이
        return new DefaultKafkaConsumerFactory<>(props, new StringDeserializer(),
            new JsonSerializer<>(Customer.class, false));
    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory<String, Customer> customerKafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, Customer> factory = new ConcurrentKafkaListenerContainerFactory<>();
        //ENABLE_AUTO_COMMIT_CONFIG= false로 지정했을때, 어떻게 commit할지를 지정한다.
        factory.getContainerProperties().setAckMode(ContainerProperties.AckMode.MANUAL_IMMEDIATE); // 즉각적으로 ack 요청한다.
        factory.setConsumerFactory(customerConsumerFactory());
        return factory;
    }
}
```

5. CQRS 구축 실습

11) CustomerSubscriber

```
@Component
public class CustomerSubscriber {

    @Autowired
    private InquiryService inquiryService;
    private final Logger LOGGER = LoggerFactory.getLogger(CustomerSubscriber.class);

    @KafkaListener(topics = "create-customer", containerFactory = "customerKafkaListenerContainerFactory")
    public void creatingCustomerListener(Customer customer, Acknowledgment ack) {
        LOGGER.info("Recieved creating customer message: " + customer.getUserid());
        try {
            /*고객 상세 조회 : 고객 데이터 등록*/
            inquiryService.insertCustomer(customer);

            ack.acknowledge();
        } catch (Exception e) {
            String msg = " 고객 데이터 생성 또는 고객 이력 생성에 문제가 발생했습니다.";
            LOGGER.error(customer.getUserid() + msg,e);
        }
    }
}
```

5. CQRS 구축 실습

12) Customer 서비스 수정

```
@Configuration
public class KafkaPublisherConfig {

    @Bean
    public ProducerFactory<String, Customer> pingProducerFactory() {
        Map<String, Object> configProps = new HashMap<>();
        configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, Customer> pingKafkaTemplate() {
        return new KafkaTemplate<>(pingProducerFactory());
    }
}
```

```
@Component
public class CustomerProducer {

    @Autowired
    private KafkaTemplate<String, Customer> customerKafkaTemplate;

    public void sendCreatingCustomerMessage(Customer customer) {
        ListenableFuture<SendResult<String, Customer>> future =
            customerKafkaTemplate.send("create-customer", customer);
    }
}
```

5. CQRS 구축 실습

```
@Service("customerService")
public class CustomerServiceImpl implements CustomerService {

    @Autowired
    CustomerRepository customerRepository;

    @Autowired
    OrderFeignClient orderFeignClient;

    //CQRS
    @Autowired
    CustomerProducer customerProducer;

    @Override
    public int insertCustomer(Customer customer) throws Exception {

        int num = customerRepository.insertCustomer(customer);
        //CQRS
        customerProducer.sendCreatingCustomerMessage(customer);

        return num;
    }
}
```

5. CQRS 구축 실습

13) 실행 및 Swagger UI 요청

<http://localhost:8076/ecommerce/customer/swagger-ui.html>

customer-controller Customer Controller

GET /detail/rest/customers/{userid} 고객상세조회

POST /rest/customer 고객 등록

고객 등록

Parameters

Name	Description
customer * required (body)	customer Example Value Model <pre>{ "pwd": "1234", "name": "이순신", "email": "lee@korea.com" }</pre>

Query Editor		Query History			Scratch Pad	
<pre>1 SELECT userid, pwd, name, email, create_date 2 FROM public.msa_customer_cqrs;</pre>						
Data Output						
Explain						
Messages						
Notifications						
userid	pwd	name	email	create_date		
[PK] character varying (50)	character varying (10)	character varying (20)	character varying (20)	date		
1 1111	1234	홍길동	hong@korea.com	2021-05-19		
2 505ceca8-01ce-4823-8890-465dca9320af	1234	이순신	lee@korea.com	2021-05-19		

계속해서
Order 서비스까지
작업 완료한다.

감사합니다.