

MSA 개발 가이드

Tags	2020 DT 교육
상태	진행
시작	@Aug 23, 2020
종료	@Aug 30, 2020

개발 순서

▼ 설계

- Event storming을 통한 마이크로서비스 설계
- msaez는 코드 생성이 명확하지 않고, 생성된 코드로 진행하면 서비스 구조나 동작원리를 이해하기 어려우며, 실무에서 사용하지 않는 툴이므로 실제 도움이 될만한 방법으로 진행하는 것이 좋다고 판단함
- 따라서, event storming을 통한 설계는 msaez로, 개발은 Spring Initializr로 시작하는 수기 코딩으로 진행

▼ 환경 설정 for MacBook Air 구형 버전(2010.10)

1. Homebrew 설치

1. 맥에서 여러가지 패키지를 설치하려면 Homebrew를 설치 해야함
2. Homebrew 동작을 위해서는 XCode 10.1 이상 버전이 필요, OS 업그레이드가 필요
3. 내 맥북 에어 버전은 최신 OS 버전인 Catalina 설치 불가 → 이려면 최신 버전의 XCode도 설치할 수 없음
4. 따라서 구 버전의 CLT (Command Line Tools) 를 찾아 설치해야 함
5. 구형 CLT를 developer.apple.com/download/more 에서 찾아 다운로드 및 설치

2. httpie 설치

1. 손쉽게 REST API Call 테스트 하기 위한 도구
2. <https://httpie.org/docs#installation>

3. kafka 다운로드

1. 다운로드 후 적당한 경로에 압축 풀기
2. https://www.apache.org/dyn/closer.cgi?path=/kafka/2.3.0/kafka_2.12-2.3.0.tgz

4. 터미널에서 kafka 실행

1. 경로 이동
`cd /Users/JK_Park/Dev/Kafka/kafka_2.12-2.3.0/bin`
2. 주키퍼 실행
`./zookeeper-server-start.sh ../config/zookeeper.properties &`
3. 카프카 broker 실행
`./kafka-server-start.sh ../config/server.properties &`
4. 카프카 topic 만들기
`./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic booking`
5. 카프카 producer 실행
`./kafka-console-producer.sh --broker-list localhost:9092 --topic booking`
6. 카프카 consumer 실행
`./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic booking --from-beginning`

5. aws cli 설치

1. https://docs.aws.amazon.com/ko_kr/cli/latest/userguide/install-cliv2-mac.html

6. eksctl 설치

1. https://docs.aws.amazon.com/ko_kr/eks/latest/userguide/getting-started-eksctl.html

7. kubectl 설치

1. <https://kubernetes.io/ko/docs/tasks/tools/install-kubectl/>

8. Docker toolbox 설치

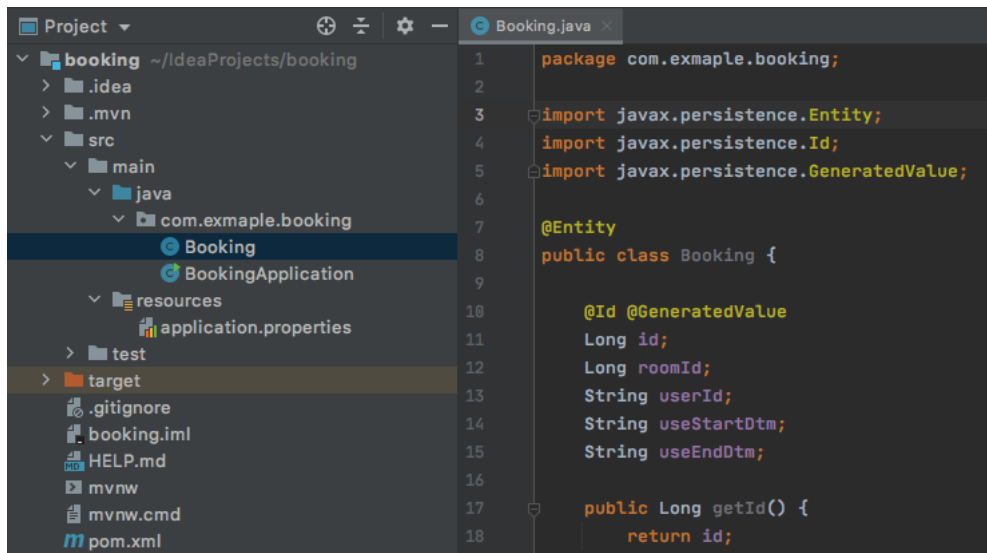
1. 내 맥북 에어 버전(2010.10) CPU는 hypervisor 지원을 하지 않아 Docker Desktop을 설치할 수 없음
2. Docker toolbox 를 설치한 후, Docker Quickstart 터미널을 통해 커맨드 라인 방식 작업을 해야 함

▼ 개발 (참고 Workflow: <https://workflowy.com/s/msa/27a0ioMCzlpV04lb#/a939fadba124>)

1. Spring Initializr (start.spring.io) 에서 서비스 세팅 후 프로젝트 다운로드

The screenshot shows the Spring Initializr configuration page. On the left, under 'Project', 'Maven Project' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', '2.3.3' is selected. Under 'Project Metadata', fields for Group (com.exmaple), Artifact (booking), Name (booking), Description (booking service), and Package name (com.exmaple.booking) are filled. Packaging is set to 'Jar' and Java version is '8'. On the right, the 'Dependencies' section shows 'Spring Data JPA' (SQL), 'H2 Database' (SQL), and 'Rest Repositories' (WEB) are added.

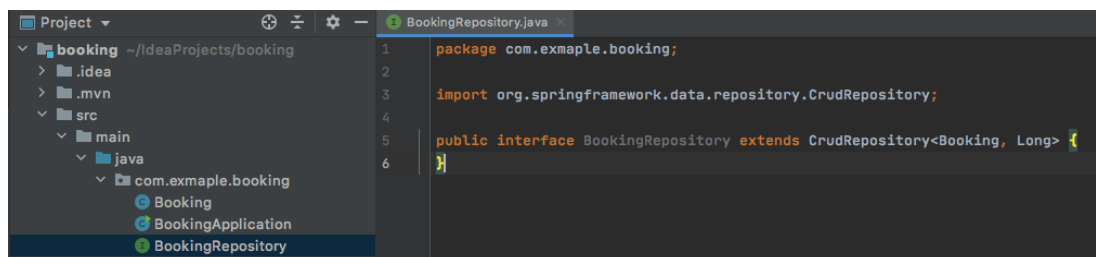
- Artifact, Name, Description 을 서비스에 맞게 수정
 - Java 8 → 원하는 버전으로 선택할 것
 - 종속성에 3개 추가: Spring Data JPA, H2 Database, Rest Repositories
 - 다운로드
2. 다운 받은 zip 파일 압축을 풀고, 해당 폴더를 IntelliJ 에서 열기
 - IntelliJ가 자동으로 Maven 프로젝트로 전환해 줌
 3. 엔티티 코딩하기
 - 서비스를 위한 속성 정의



- 서비스 패키지에 class 신규 생성
- 엔티티에 attribute 선언
- id 에 @Id, @GeneratedValue annotation 붙여주기: 자동생성되는 엔티티의 PK 값
- Getter, setter 자동 생성

4. 리파지토리 코딩하기

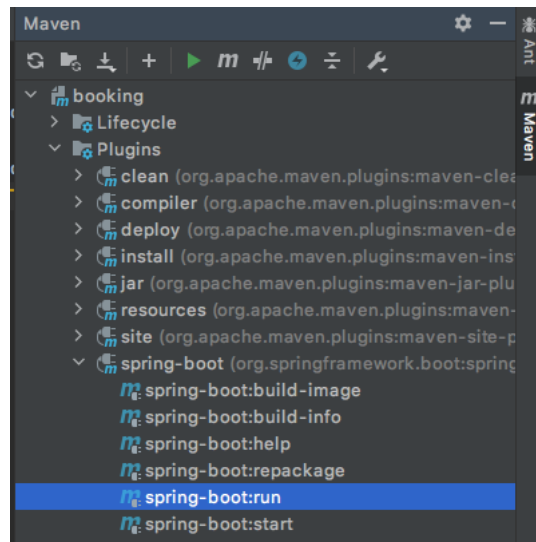
- 엔티티의 CRUD를 수행



- 서비스 패키지에 interface 신규 생성
- CurdRepository 상속

5. 엔티티, 리파지토리 테스트

- 서비스 실행



- Maven > booking > Plugins > spring-boot:run
- Httpie를 통한 REST API Call 테스트
 - http GET localhost:8080/bookings

```
JKs-MacBook-Air:~ JK_Park$ http GET localhost:8080/bookings
HTTP/1.1 200
Connection: keep-alive
Content-Type: application/hal+json
Date: Tue, 25 Aug 2020 00:19:18 GMT
Keep-Alive: timeout=60
Transfer-Encoding: chunked
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers

{
  "_embedded": {
    "bookings": []
  },
  "_links": {
    "profile": {
      "href": "http://localhost:8080/profile/bookings"
    },
    "self": {
      "href": "http://localhost:8080/bookings"
    }
  }
}
```

- http POST localhost:8080/bookings roomId=1 userId="06719" useStartDtm="20200825090000" useEndDtm="20200825100000"

```
JKs-MacBook-Air:~ JK_Park$ http POST localhost:8080/bookings roomId=1 userId="06719" "useStartDtm="20200825090000" useEndDtm="20200825100000"
HTTP/1.1 201
Connection: keep-alive
Content-Type: application/json
Date: Tue, 25 Aug 2020 00:20:20 GMT
Keep-Alive: timeout=60
Location: http://localhost:8080/bookings/1
Transfer-Encoding: chunked
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers

{
  "_links": {
    "booking": {
      "href": "http://localhost:8080/bookings/1"
    },
    "self": {
      "href": "http://localhost:8080/bookings/1"
    }
  },
  "roomId": 1,
  "useEndDtm": "20200825100000",
  "useStartDtm": "20200825090000",
  "userId": "06719"
}

JKs-MacBook-Air:~ JK_Park$
```

- http GET localhost:8080/bookings/1

```
JKs-MacBook-Air:~ JK_Park$ http GET localhost:8080/bookings/1
HTTP/1.1 200
Connection: keep-alive
Content-Type: application/hal+json
Date: Tue, 25 Aug 2020 00:20:51 GMT
Keep-Alive: timeout=60
Transfer-Encoding: chunked
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers

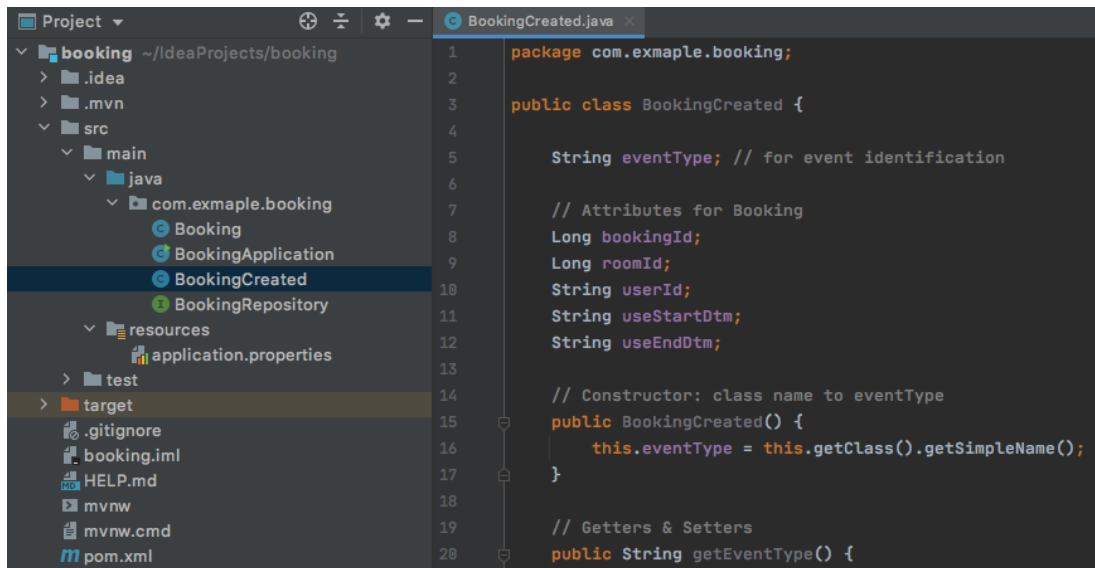
{
  "_links": {
    "booking": {
      "href": "http://localhost:8080/bookings/1"
    },
    "self": {
      "href": "http://localhost:8080/bookings/1"
    }
  },
  "roomId": 1,
  "useEndDtm": "20200825100000",
  "useStartDtm": "20200825090000",
  "userId": "06719"
}

JKs-MacBook-Air:~ JK_Park$
```

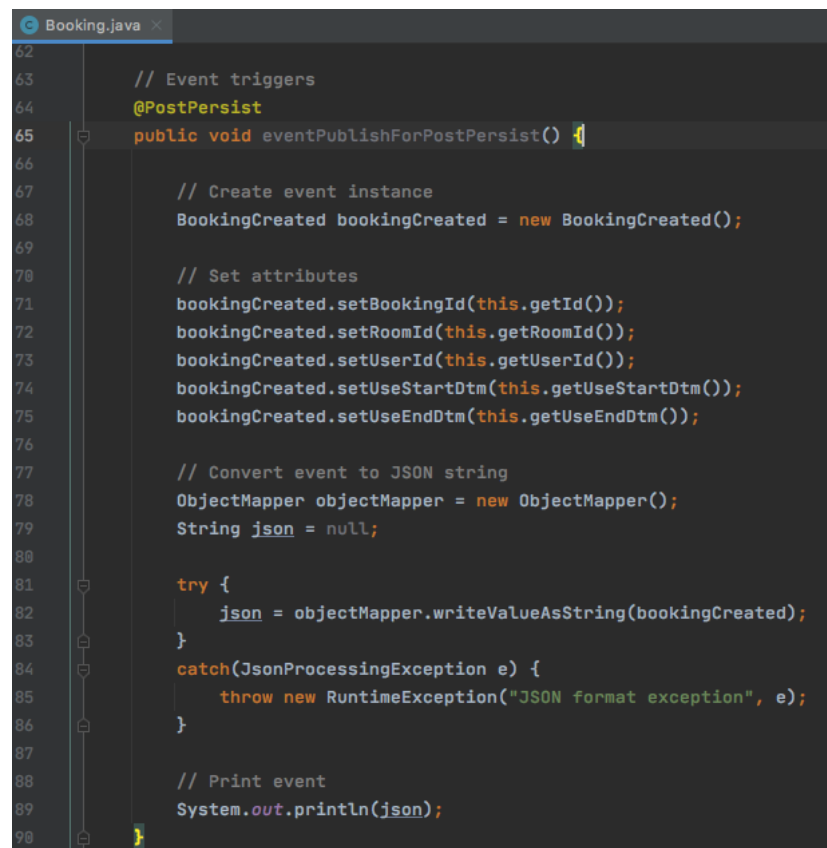
- Spring boot가 엔티티와 리파지토리 기준으로 REST API와 CRUD 로직을 자동으로 만들어 주는것 같음

6. 이벤트 코딩하기

- 이벤트 클래스 생성



- eventType 변수로 이벤트를 식별
- 서비스 엔티티의 속성들을 그대로 가져옴
- 클래스 생성시 클래스 이름(= 이벤트 이름)을 eventType 값으로 할당
- Getter 와 setter를 자동생성
- 엔티티 클래스 안에 이벤트 트리거 생성
 - 메소드 생성



- 이벤트 트리거 종류에 따라 메소드를 생성해야 하므로 메소드 이름을 잘 정해야 함

7. 이벤트 테스트

- 생성 이벤트 발생 후 콘솔 로그 (System.out.println() 실행부) 에 이벤트 결과를 정상적으로 출력하는지 확인

```

JKs-MacBook-Air:~ JK_Park$ http POST localhost:8080/bookings roomId=1 userId="06719"
" useStartDtm="20200825090000" useEndDtm="20200825100000"
HTTP/1.1 201
Connection: keep-alive
Content-Type: application/json
Date: Tue, 25 Aug 2020 01:22:45 GMT
Keep-Alive: timeout=60
Location: http://localhost:8080/bookings/1
Transfer-Encoding: chunked
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers

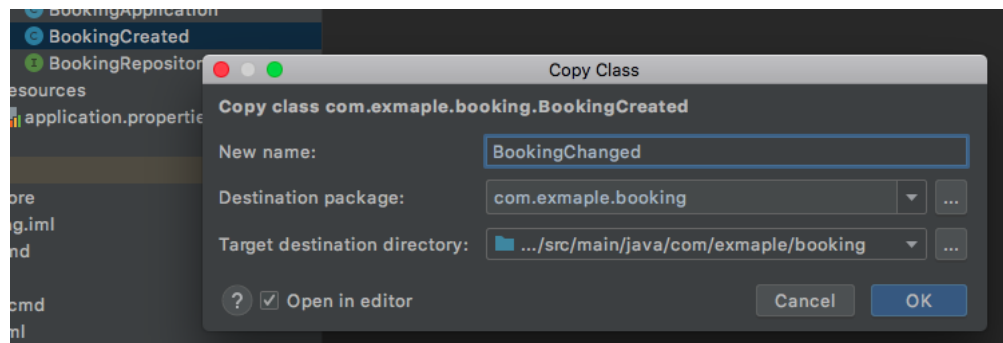
{
  "_links": {
    "booking": {
      "href": "http://localhost:8080/bookings/1"
    },
    "self": {
      "href": "http://localhost:8080/bookings/1"
    }
  },
  "roomId": 1,
  "useEndDtm": "20200825100000",
  "useStartDtm": "20200825090000",
  "userId": "06719"
}
2020-08-25 10:22:37.672 INFO 52178 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat]
JKs-MacBook-Air:~ JK_Park$ : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-08-25 10:22:37.673 INFO 52178 --- [nio-8080-exec-1] o.s.web.servlet
.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2020-08-25 10:22:37.711 INFO 52178 --- [nio-8080-exec-1] o.s.web.servlet
.DispatcherServlet : Completed initialization in 38 ms
{"eventType":"BookingCreated","bookingId":1,"roomId":1,"userId":"06719",
"useStartDtm":"20200825090000","useEndDtm":"20200825100000"}

```

→ 로그 출력 성공:

```
{ "eventType": "BookingCreated", "bookingId": 1, "roomId": 1, "userId": "06719", "useStartDtm": "20200825090000", "useEndDtm": "20200825100000" }
```

- 테스트가 성공했다면 같은 방식으로 다른 이벤트도 코딩, 테스트
 - 최초 만든 이벤트 클래스를 Copy & Paste (Ctrl + C, V) 하면 빠르게 생성 가능함. 왜냐하면 내부 로직을 보면 바꿀 것이 없음.



- 엔티티에 이벤트 트리거 메소드 생성
 - 역시 기존 메소드를 복사해서 생성하면 되며, 추가적인 로직이 없을 경우 IntelliJ의 Refactor 기능을 이용하여 이벤트 클래스 변수명을 바꿔주면 역시 빠르게 생성 가능함.

```

89      @PostUpdate
90      public void eventPublishForPostUpdate() {...}
116
117      @PostRemove
118      public void eventPublishForPostRemove() {...}

```

8. 프로젝트에 kafka 라이브러리 추가

- pom.xml 에 kafka dependency 추가

```

50
51     <dependency>
52         <groupId>org.springframework.cloud</groupId>
53         <artifactId>spring-cloud-starter-stream-kafka</artifactId>
54     </dependency>
55
56     <dependency>
57         <groupId>org.springframework.boot</groupId>
58         <artifactId>spring-boot-starter-actuator</artifactId>
59     </dependency>

```

- dependencyManagement 추가

```

63     <dependencyManagement>
64         <dependencies>
65             <dependency>
66                 <groupId>org.springframework.cloud</groupId>
67                 <artifactId>spring-cloud-dependencies</artifactId>
68                 <version>${spring-cloud.version}</version>
69                 <type>pom</type>
70                 <scope>import</scope>
71             </dependency>
72         </dependencies>
73     </dependencyManagement>

```

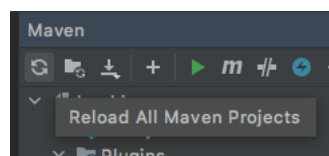
- 스프링 클라우드 버전 추가

```

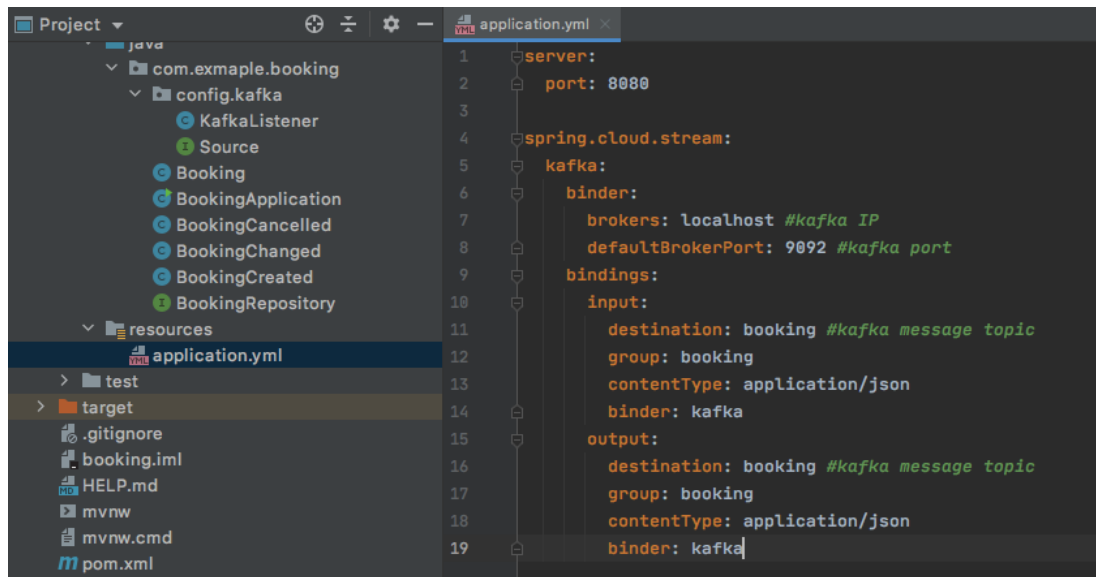
17     <properties>
18         <java.version>8</java.version>
19         <spring-cloud.version>Hoxton.SR5</spring-cloud.version>
20     </properties>

```

- Maven 프로젝트 리로드

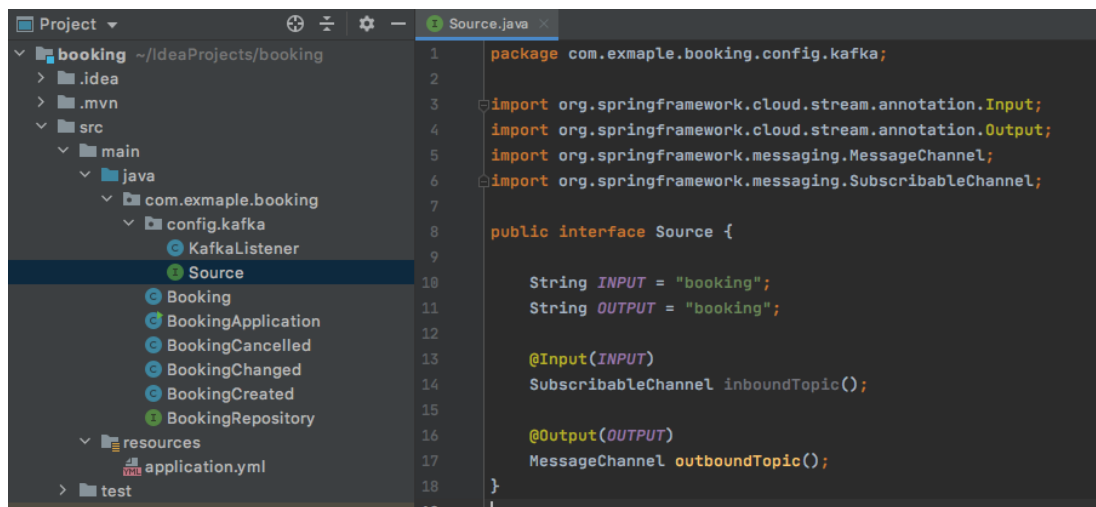


- application.yml 파일에 카프카 세팅 추가



9. Kafka pub/sub 코딩하기

- 참조
 - https://docs.spring.io/spring-cloud-stream/docs/Brooklyn.RELEASE/reference/html/_getting_started.html
- kafka 메시지 채널 생성을 위해 interface 하나를 만들어 줘야 함



- `@EnableBiding` annotation을 추가하여 위의 interface를 참조하는 메시지 채널을 생성

```

BookingApplication.java x
1 package com.exmaple.booking;
2
3 import com.exmaple.booking.config.kafka.Source;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
6 import org.springframework.cloud.stream.annotation.EnableBinding;
7 import org.springframework.context.ApplicationContext;
8
9 @SpringBootApplication
10 @EnableBinding(Source.class)
11 public class BookingApplication {
12
13     public static ApplicationContext applicationContext;
14
15     public static void main(String[] args) {
16         applicationContext = SpringApplication.run(BookingApplication.class, args);
17     }
18
19 }

```

- 이벤트 트리거 메소드 안에 kafka 로 메시지를 전송하는 코드 추가 (91 ~ 98 라인)

```

Booking.java x
87 catch(JsonProcessingException e) {
88     throw new RuntimeException("JSON format exception", e);
89 }
90
91 // Send message to kafka
92 Source processor = BookingApplication.applicationContext.getBean(Source.class);
93 MessageChannel outputChannel = processor.outboundTopic();
94
95 outputChannel.send(MessageBuilder
96     .withPayload(json)
97     .setHeader(MessageHeaders.CONTENT_TYPE, MimeTypeUtils.APPLICATION_JSON)
98     .build());
99 }

```

- 여기서도 Source interface의 메시지 채널을 사용

10. Kafka pub/sub 테스트

- 터미널에서 kafka consumer를 켜 놓고, REST API 호출 테스트 수행

```
bin — bash — 117x33
JKs-MacBook-Air:bin JK_Park$ http POST localhost:8080/bookings roomId=1 userId="06719" useStartDtm="20200825090000" useEndDtm="20200825100000"
HTTP/1.1 201
Connection: keep-alive
Content-Type: application/json
Date: Tue, 25 Aug 2020 05:32:49 GMT
Keep-Alive: timeout=60
Location: http://localhost:8080/bookings/5
Transfer-Encoding: chunked
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
{"_links": {"booking": {"href": "http://localhost:8080/bookings/5"}}, "roomId": 1, "useEndDtm": "20200825100000", "useStartDtm": "20200825090000", "userId": "06719"}
JKs-MacBook-Air:bin JK_Park$

bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=10
JKs-MacBook-Air:bin JK_Park$ ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic booking
[2020-08-25 14:32:32,209] INFO [GroupCoordinator 0]: Preparing to rebalance group console-consumer-20327 in state PreparingRebalance with old generation 0 (__consumer_offsets-35) (reason: Adding new member consumer-1-800ed5d7-feb7-46e5-973c-9b0b885ec788 with group instanceid None) (kafka.coordinator.group.GroupCoordinator)
[2020-08-25 14:32:32,212] INFO [GroupCoordinator 0]: Stabilized group console-consumer-20327 generation 1 (__consumer_offsets-35) (kafka.coordinator.group.GroupCoordinator)
[2020-08-25 14:32:32,223] INFO [GroupCoordinator 0]: Assignment received from leader for group console-consumer-20327 for generation 1 (kafka.coordinator.group.GroupCoordinator)
{"eventType": "BookingCreated", "bookingId": 5, "roomId": 1, "userId": "06719", "useStartDtm": "20200825090000", "useEndDtm": "20200825100000"}
kafka.tools.ConsumerToolRunner: Kafka version: 2.5.1
kafka.clients.Metadata: [Producer clientId=producer-2] Cluster ID:
kafka.tools.ConsumerToolRunner: Kafka commitId: 0efa8fb0f4c73d92
kafka.tools.ConsumerToolRunner: Kafka startTimestampMs: 1598332458877
```

- 테스트 성공 후 다른 이벤트에 대해서도 pub/sub 코딩을 하면 됨
 - 추가한 코딩에 이벤트 관련 내용이 없으므로 복붙하면 됨

▼ 배포

1. 코드 레파지토리 생성
2. CodeBuild 생성
3. CodeBuild - 코드 레파지토리 연결
4. CodeBuild - ECR 연결