

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

9. Cloud9 환경 설정

- 코드 에디터가 달려 있는 리눅스 환경이라 생각하면 되고, 대부분의 명령은 커맨드라인 기반임
- 로컬 환경과 동일하다고 생각하고 코딩하면 됨

1. AWS Console에서 환경 생성

Step 1
Name environment

Step 2
Configure settings

Step 3
Review

Name environment

Environment name and description

Name
The name needs to be unique per user. You can update it at any time.

Limit: 60 characters

Description - Optional
This will appear on your environment's card in your dashboard. You can update it at any time.

- 이름과 설명 입력

Environment settings

Environment type [Info](#)

Run your environment in a new EC2 instance or an existing server. With EC2 instances, you can connect directly via SSH or connect via AWS Systems Manager (without opening inbound ports).

- ☒ Create a new EC2 instance for environment (direct access)
Launch a new instance in this region that your environment can access directly via SSH.
- ☐ Create a new no-ingress EC2 instance for environment (access via Systems Manager)
Launch a new instance in this region that your environment can access through Systems Manager.
- ☐ Create and run in remote server (SSH connection)
Configure the secure connection to the remote server for your environment.

Instance type

- ☒ t2.micro (1 GiB RAM + 1 vCPU)
Free-tier eligible. Ideal for educational users and exploration.
- ☐ t3.small (2 GiB RAM + 2 vCPU)
Recommended for small-sized web projects.
- ☐ m5.large (8 GiB RAM + 2 vCPU)
Recommended for production and general-purpose development.
- ☐ Other instance type
Select an instance type.

t3.nano

Platform

- ☐ Amazon Linux
- ☒ Ubuntu Server 18.04 LTS

Cost-saving setting

Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary chargeback settings of half an hour of no activity to maximize savings.

After 30 minutes (default)

- Platform은 익숙한 Ubuntu 18.04 선택
2. Maven 이 없으므로 설치 해줌
 - 터미널에 명령어 입력 (코딩 기능 말고는 터미널을 써야 함)
 - `sudo apt install maven`

```
park108:~/environment/msa-booking (master) $ sudo apt install maven
```

3. GitHub 리포지토리에서 소스를 클론해서 받아옴

```
park108:~/environment $ git clone https://github.com/park108/msa-booking
Cloning into 'msa-booking'...
remote: Enumerating objects: 68, done.
remote: Counting objects: 100% (68/68), done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 68 (delta 21), reused 60 (delta 13), pack-reused 0
Unpacking objects: 100% (68/68), done.
```

- 터미널에 명령어 입력
 - `git clone {github 리포지토리 URL}`
4. 코드 수정 후 github로 push

```

bash - "ip-172-31-3-174" x
park108:~/environment/msa-booking (master) $ git push -u origin master
Username for 'https://github.com/park108/msa-booking': park108@gmail.com
Password for 'https://park108@gmail.com@github.com/park108/msa-booking':
Counting objects: 3, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 304 bytes | 304.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/park108/msa-booking
608522a..533a8af master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

```

- 터미널에 명령어 입력
- git push -u origin master

5. github 리포지토리가 CodeBuild 와 연결되어 있으면 빌드, 테스트, ECR 배포가 자동으로 수행됨

빌드 기록							
		빌드 중지		아티팩트 보기	로그 보기	빌드 삭제	빌드 재시도
		< 1 >					
<input type="checkbox"/>	빌드 실행	상태	빌드 번호	소스 버전	제출자	기간	완료됨
<input type="checkbox"/>	msa-booking:e40550c2-b1cc-4bab-9af7-b9191371386b	🟢 성공함	2	533a8af9586279eebec01649338d692a31b50be8	GitHub-Hookshot/d696b2a	1분 12초	1분 전
<input type="checkbox"/>	이미지 태그	이미지 URI				푸시 위치	
<input type="checkbox"/>	latest	351904566406.dkr.ecr.us-east-1.amazonaws.com/msa-booking:latest				20.08.26. 오후 02:22	
<input type="checkbox"/>	<untagged>	351904566406.dkr.ecr.us-east-1.amazonaws.com/msa-booking@sha256:fb7096bc44f7249b5ff78c77000a8936737770156fe5dae64529c4755e6a498				20.08.26. 오후 09:39	

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

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

Project
☒ Maven Project
☐ Gradle Project

Language
☒ Java ☐ Kotlin
☐ Groovy

Spring Boot
☐ 2.4.0 (SNAPSHOT) ☐ 2.4.0 (M2)
☐ 2.3.4 (SNAPSHOT) ☒ 2.3.3 ☐ 2.2.10 (SNAPSHOT)
☐ 2.2.9 ☐ 2.1.17 (SNAPSHOT) ☐ 2.1.16

Project Metadata

Group	com.exmaple
Artifact	booking
Name	booking
Description	booking service
Package name	com.exmaple.booking
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 14 <input type="radio"/> 11 <input checked="" type="radio"/> 8

Dependencies ADD DEPENDENCIES... + B

Spring Data JPA SOL
 Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

H2 Database SOL
 Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Rest Repositories WEB
 Exposing Spring Data repositories over REST via Spring Data REST.

- 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

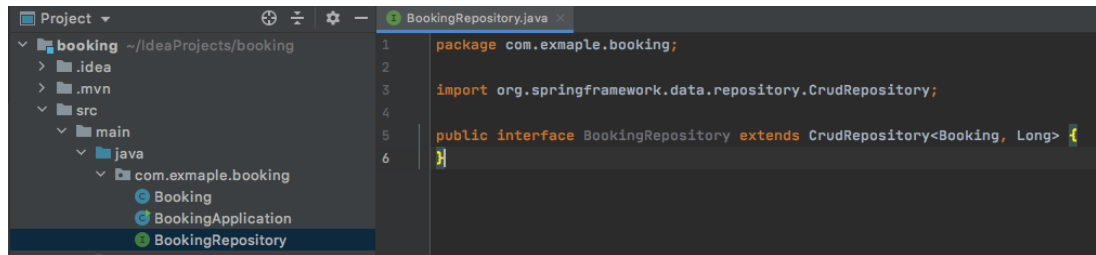
```

1 package com.exmaple.booking;
2
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5 import javax.persistence.GeneratedValue;
6
7 @Entity
8 public class Booking {
9
10     @Id @GeneratedValue
11     Long id;
12     Long roomId;
13     String userId;
14     String useStartDtm;
15     String useEndDtm;
16
17     public Long getId() {
18         return id;
19     }
20 }
  
```

- 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

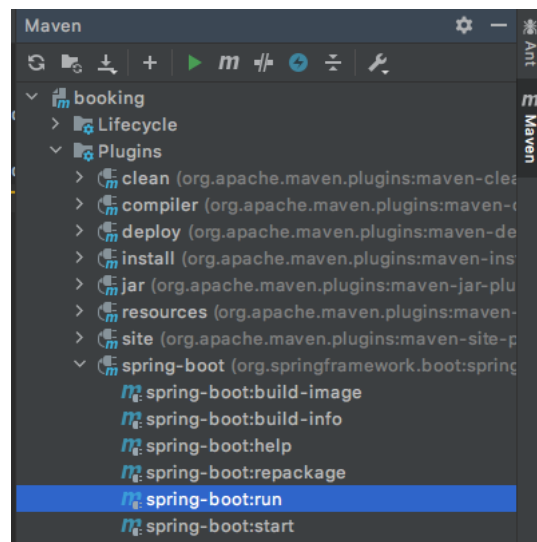
- 엔티티의 CRUD를 수행



- 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"
    }
  }
}

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

- 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

```

package com.exmaple.booking;

public class BookingCreated {

    String eventType; // for event identification

    // Attributes for Booking
    Long bookingId;
    Long roomId;
    String userId;
    String useStartDtm;
    String useEndDtm;

    // Constructor: class name to eventType
    public BookingCreated() {
        this.eventType = this.getClass().getSimpleName();
    }

    // Getters & Setters
    public String getEventType() {

```

- 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


```

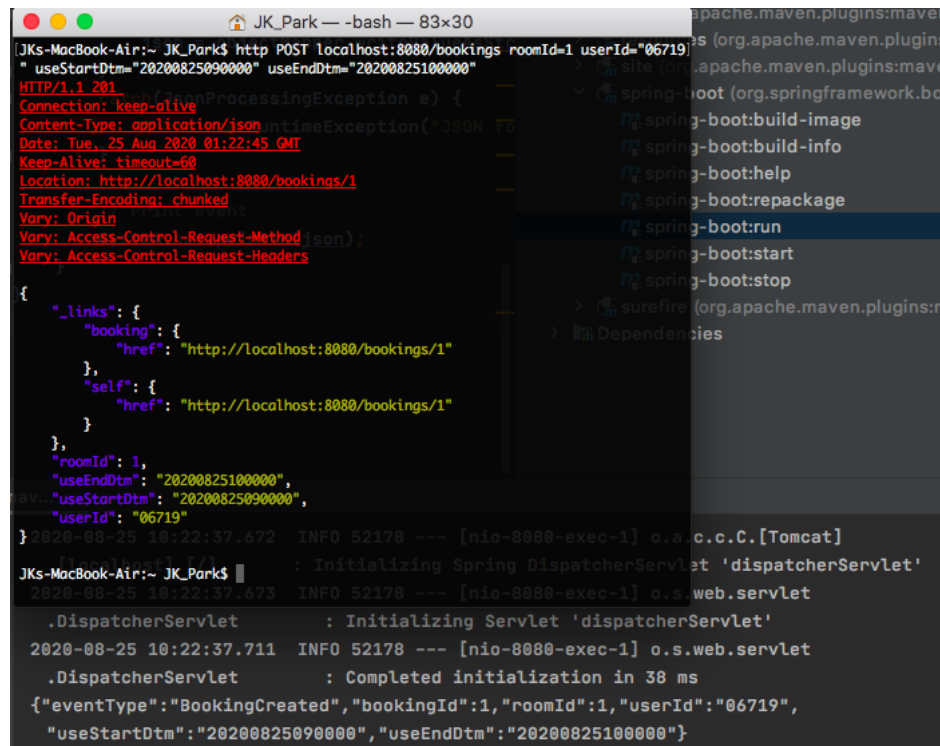
62
63 // Event triggers
64 @PostPersist
65 public void eventPublishForPostPersist() {
66
67     // Create event instance
68     BookingCreated bookingCreated = new BookingCreated();
69
70     // Set attributes
71     bookingCreated.setBookingId(this.getId());
72     bookingCreated.setRoomId(this.getRoomId());
73     bookingCreated.setUserId(this.getUserId());
74     bookingCreated.setUseStartDtm(this.getUseStartDtm());
75     bookingCreated.setUseEndDtm(this.getUseEndDtm());
76
77     // Convert event to JSON string
78     ObjectMapper objectMapper = new ObjectMapper();
79     String json = null;
80
81     try {
82         json = objectMapper.writeValueAsString(bookingCreated);
83     }
84     catch(JsonProcessingException e) {
85         throw new RuntimeException("JSON format exception", e);
86     }
87
88     // Print event
89     System.out.println(json);
90 }

```

- 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



```

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

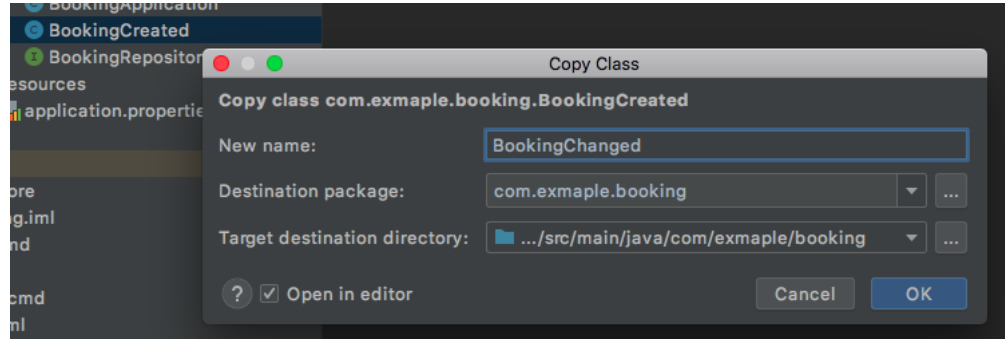
{
  "_links": {
    "booking": {
      "href": "http://localhost:8080/bookings/1"
    },
    "self": {
      "href": "http://localhost:8080/bookings/1"
    }
  },
  "roomId": 1,
  "useEndDtm": "20200825100000",
  "useStartDtm": "20200825090000",
  "userId": "06719"
}
2020-08-25 10:22:37.672 INFO 52178 --- [nio-8080-exec-1] o.s.c.c.C.[Tomcat]
: 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","t
```

- 테스트가 성공했다면 같은 방식으로 다른 이벤트도 코딩, 테스트
 - 최초 만든 이벤트 클래스를 Copy & Paste (Ctrl + C, V) 하면 빠르게 생성 가능함. 왜냐면 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

```
m pom.xml (booking) x
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

```
m pom.xml (booking) x
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>
```

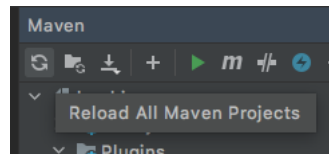
- 스프링 클라우드 버전 추가

```

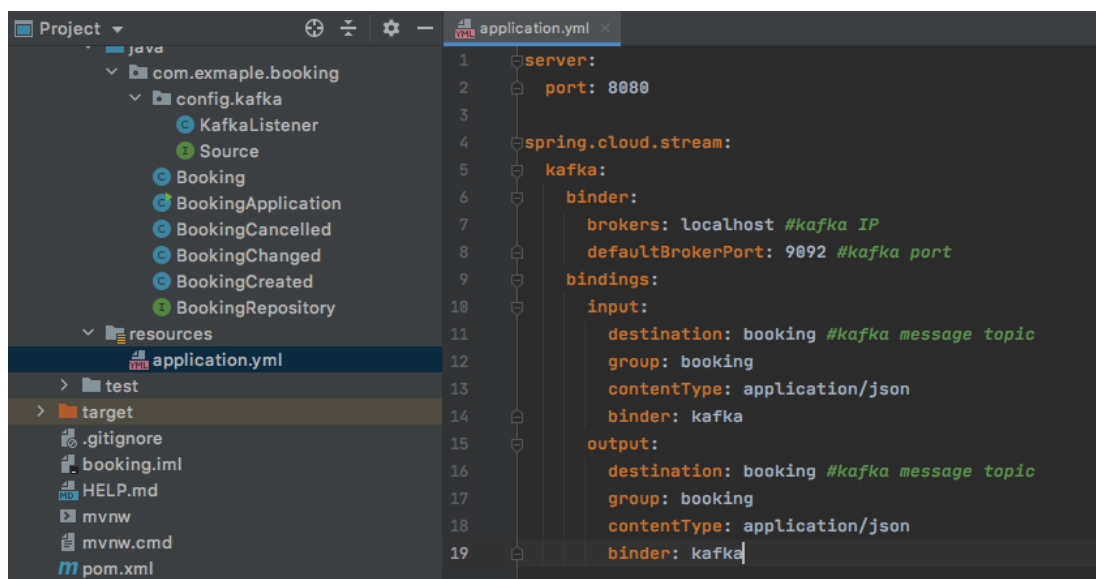
17     <properties>
18         <java.version>8</java.version>
19         <spring-cloud.version>Hoxton.SR5</spring-cloud.version>
20     </properties>

```

- 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

```

1 package com.exmaple.booking.config.kafka;
2
3 import org.springframework.cloud.stream.annotation.Input;
4 import org.springframework.cloud.stream.annotation.Output;
5 import org.springframework.messaging.MessageChannel;
6 import org.springframework.messaging.SubscribableChannel;
7
8 public interface Source {
9
10     String INPUT = "booking";
11     String OUTPUT = "booking";
12
13     @Input(INPUT)
14     SubscribableChannel inboundTopic();
15
16     @Output(OUTPUT)
17     MessageChannel outboundTopic();
18 }

```

- @EnableBiding annotation을 추가하여 위의 interface를 참조하는 메시지 채널을 생성

```

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 }

```

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

```

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, "useStartDtm": "20200825090000", "useEndDtm": "20200825100000", "userId": "06719"}

JKs-MacBook-Air:bin JK_Park$

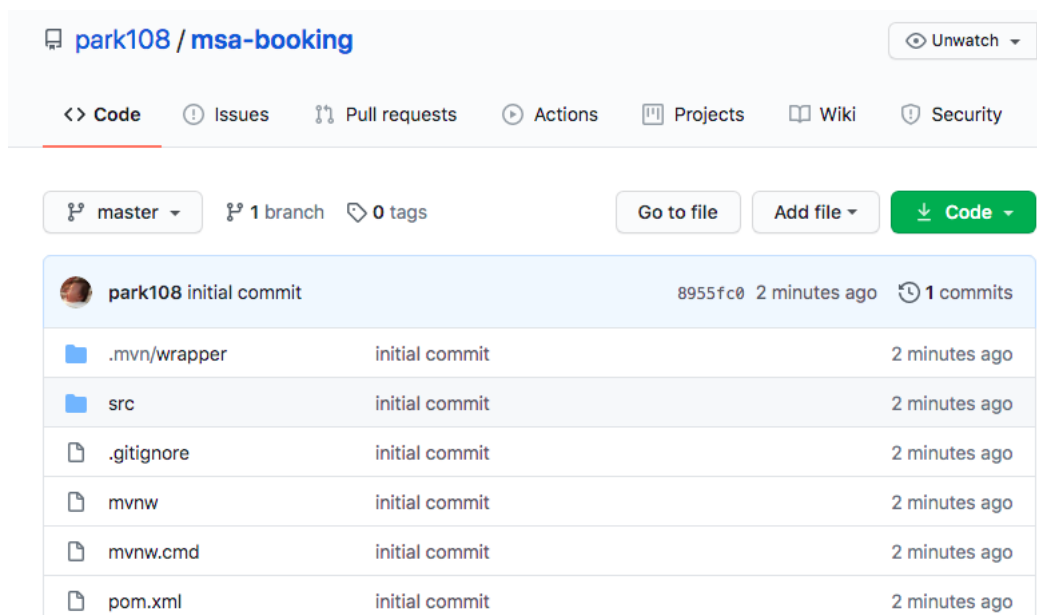
bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=10
JKs-MacBook-Air:bin JK_Park$ ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic booking
[2020-08-25 14:32:32,209] INFO [GroupCoordinator 0]: Preparing to rebalance group console-consumer-20327 in state PreparingRebalance with old generation 0 (__consumer_offsets-35) (reason: Adding new member consumer-1-800ed5d7-feb7-46e5-973c-9b0b888ec788 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.ConsumerInfoParser : Kafka version: 2.5.1
kafka.clients.Metadata : [Producer clientId=producer-2] Cluster ID:
common.utils.AppInfoParser : Kafka commitId: 0efa8fb0f4c73d92
common.utils.AppInfoParser : Kafka startTimeMs: 1598332458877

```

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

▼ 배포: Github → CodeBuild → ECR 자동 빌드 구성

1. github.com 코드 리포지토리 생성 및 로컬 소스 push

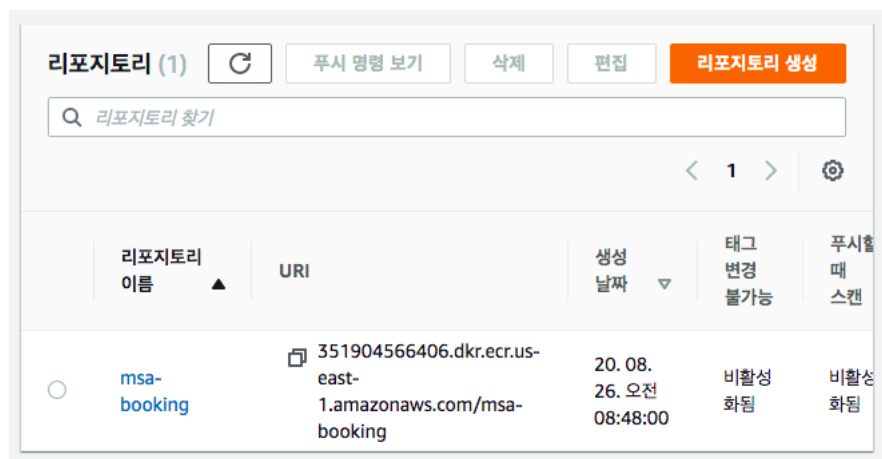


2. Amazon ECR 리포지토리 생성

- 리포지토리 이름을 넣고 생성



- 만들고 URI를 copy 해 둬



리포지토리 이름	URI	생성 날짜	태그 변경 불가능	푸시할 때 스캔
msa-booking	351904566406.dkr.ecr.us-east-1.amazonaws.com/msa-booking	20. 08. 26. 오전 08:48:00	비활성화됨	비활성화됨

3. 로컬에서 도커 이미지를 생성하여 초기 적재

- 터미널에서 프로젝트 디렉토리로 이동하여 패키징 수행
 - mvn clean
 - mvn package
- Dockerfile 생성



```
1 FROM openjdk:8u212-jdk-alpine
2 COPY target/*SNAPSHOT.jar app.jar
3 EXPOSE 8080
4 ENTRYPOINT ["java", "-Xmx4096M", "-Djava.security.edg=file:/dev/.urandom", "-jar", "/app.jar", "--spring.profiles.active=docker"]
```

- 프로젝트 root 에 생성
- 도커 이미지 생성
 - docker build -t {복사한 ECR repository URI}:{지정할 태그} .

```
JKs-MacBook-Air:booking JK_Park$ docker build -t 351904566406.dkr.ecr.us-east-1.amazonaws.com/msa-booking:latest .
Sending build context to Docker daemon 58.64MB
Step 1/4 : FROM openjdk:8u212-jdk-alpine
--> a3562a0b991
Step 2/4 : COPY target/*SNAPSHOT.jar app.jar
--> 5c6c3e81611
Step 3/4 : EXPOSE 8080
--> Running in cee71cab7603
Removing intermediate container cee71cab7603
--> 1f48cf36a29e
Step 4/4 : ENTRYPOINT ["java", "-Xmx400M", "-Djava.security.edg=file:/dev/./urandom", "-jar", "/app.jar", "--spring.profiles.active=docker"]
--> Running in 4e6eecd8277f
Removing intermediate container 4e6eecd8277f
--> 88b76261d8d1
Successfully built 88b76261d8d1
Successfully tagged 351904566406.dkr.ecr.us-east-1.amazonaws.com/msa-booking:latest
```

- 예시: `docker build -t 351904566406.dkr.ecr.us-east-1.amazonaws.com/msa-booking:latest .`
- 터미널에서 ECR 접속
 - `aws ecr get-login-password --region {리전} | docker login --username AWS --password-stdin {복사한 ECR repository URI}`

```
JKs-MacBook-Air:~$ docker JK_Park$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 351904566406.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /Users/JK_Park/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

- 예시: `aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 351904566406.dkr.ecr.us-east-1.amazonaws.com`
- 접속이 안될 경우, 도커 설정 파일을 수정
 - `~/docker/config.json`
 - "credsStore" : "desktop", 키/값 삭제
- 이미지 푸시
 - `docker push {복사한 ECR repository URI}:{태그}`

```
JKs-MacBook-Air:booking JK_Park$ docker push 351904566406.dkr.ecr.us-east-1.amazonaws.com/msa-booking:latest
The push refers to repository [351904566406.dkr.ecr.us-east-1.amazonaws.com/msa-booking]
0cf002fbff1f: Pushing [----->] 36.18MB/58.01MB
ceaf9e1ebef5: Pushing [----->] 38MB/99.29MB
9b9b7f3d56a0: Pushed
f1b5933fe4b5: Pushed
```

- 예시: `docker push 351904566406.dkr.ecr.us-east-1.amazonaws.com/msa-booking:latest`
- #### 4. AWS CodeBuild 생성
- 이름 및 설명 입력

프로젝트 구성

프로젝트 이름

msa-booking

프로젝트 이름은 2~255자여야 합니다. 글자(A-Z 및 a-z), 숫자(0-9) 및 특수 문자(- 및 _)를

설명 - 선택 사항

Booking service

- [Github.com](#) 리포지토리 연결

소스

소스 1 - 기본

소스 공급자

GitHub

리포지토리

☐ 퍼블릭 리포지토리

☒ 내 GitHub 계정의 리포지토리

GitHub 리포지토리

https://github.com/park108/msa-booking.git

https://github.com/<user-name>/<repository-name>

- Webhook 활성화

기본 소스 Webhook 이벤트 정보

Webhook - 선택 사항

☒ 코드 변경이 이 리포지토리에 푸시될 때마다 다시 빌드

빌드 유형

☒ 단일 빌드
단일 빌드를 트리거합니다.

☐ 배치 빌드
여러 빌드

Webhook 이벤트 필터 그룹 1

이벤트 유형

한 개 이상의 Webhook 이벤트 필터 그룹을 추가하여 새 빌드를 트리거하는 이벤트를 지정합니다. v
경이 리포지토리에 푸시될 때마다 새 빌드가 트리거됩니다.

PUSH X

- Github 이벤트가 CodeBuild를 트리거 함
- 빌드를 트리거 할 이벤트 유형 선택: PUSH
- 빌드 환경 설정

환경

환경 이미지

☒ 관리형 이미지
AWS CodeBuild에서 관리하는 이미지 사용

☐ 사용자 지정 이미지

운영 체제

Ubuntu

이제 프로그래밍 언어 런타임은 Ubuntu 18.04의 표준 이미지에 포함됩니다. 자세한 내용은 CodeBuild에서 제공하는 Docker Hub 페이지를 참조하십시오.

런타임

Standard

이미지

aws/codebuild/standard:4.0

이미지 버전

이 런타임 버전에 항상 최신 이미지 사용

환경 유형

Linux

- Ubuntu, Standard, aws/codebuild/standard:4.0 선택
- 권한 설정

권한이 있음

☒ 도커 이미지를 빌드하거나 빌드의 권한을 승격하려면 이 플래그를 활성화합니다.

서비스 역할

☒ 새 서비스 역할
계정에 서비스 역할 생성

☐ 기존 서비스 역할
계정에서 기존 서비스 역할 선택

역할 이름

codebuild-msa-booking-service-role

서비스 역할 이름 입력

- 도커 이미지 빌드를 위해서는 "권한이 있음" 을 체크해야 함
- 새 서비스 역할 생성 (이전에 만든것을 사용하고 싶다면 기존 서비스 역할 선택)
- 추가구성 > 환경 변수 설정

환경 변수 이름	값
AWS_DEFAULT_REGION	us-east-1
AWS_ACCOUNT_ID	351904566406
IMAGE_REPO_NAME	msa-booking
IMAGE_TAG	latest

- 여기서 설정한 값들이 향후 CodeBuild 명령어(buildspec.yml) 에서 사용됨
- 개인별 리전, 도커 이미지 이름에 맞춰서 입력할 것
- Buildspec 사용으로 체크

Buildspec

빌드 사양

☒ **buildspec 파일 사용**
YAML 형식의 buildspec 파일에 빌드 명령 저장

Buildspec 이름 - 선택 사항

기본적으로 CodeBuild는 소스 코드 루트 디렉터리에서 buildspec.yml 파일을 찾습

- CodeBuild 에서 어떻게 빌드할지 커맨드를 입력하는 설정 파일
 - 위에서 설정한 환경 변수들을 참조하여 실행됨
5. CodeBuild 서비스 역할의 정책변경
- ECR에 도커 이미지를 배포할 것이므로 관련 권한이 필요함
 - Concept
 - 역할 Role: 사용자, 서비스등이 수행하는 역할을 정의
 - 정책 Policy: 권한 정의
 - 일반적으로 역할에 정책을 연결하여 권한 제어 수행
 - IAM > 역할에서 앞서 생성한 CodeBuild의 역할을 찾아 클릭해 보면, 연결된 정책이 있음

Identity and Access Management(IAM)

- 대시보드
- ▼ 액세스 관리
 - 그룹
 - 사용자
 - 역할**
 - 정책
 - 자격 증명 공급자
 - 계정 설정
- ▼ 보고서 액세스
 - 액세스 분석기
 - 아카이브 규칙
 - 분석기
 - 설정
 - 자격 증명 보고서

역할 > codebuild-msa-booking-service-role

요약

역할 ARN	arn:aws:iam::351904566
역할 설명	편집
인스턴스 프로파일 ARN	
경로	/service-role/
생성 시간	2020-08-26 09:08 UTC+
마지막 활동	추적 기간에 액세스되지 않음
최대 세션 지속 시간	1 시간 편집

권한

신뢰 관계

태그

액세스 관리자

세션 취소

▼ Permissions policies (1 정책이 적용됨)

[정책 연결](#)

정책 이름 ▼

▶ [CodeBuildBasePolicy-msa-booking-us-east-1](#)

- 예시
 - 역할: codebuild-msa-booking-service-role
 - 정책: CodeBuildBasePolicy-msa-booking-us-east-1
- 정책을 클릭 > 정책 편집 버튼 클릭 > JSON 탭으로 이동해서 ECR 관련 권한을 추가

```

42 {
43   "Effect": "Allow",
44   "Action": [
45     "ecr:BatchCheckLayerAvailability",
46     "ecr:CompleteLayerUpload",
47     "ecr:GetAuthorizationToken",
48     "ecr:InitiateLayerUpload",
49     "ecr:PutImage",
50     "ecr:UploadLayerPart"
51   ],
52   "Resource": "*"
53 }
54

```

- 정책 검토 > 변경 내역을 저장
6. buildspec.yml 작성
- 프로젝트 루트 디렉토리에 buildspec.yml 파일 생성

```

1 version: 0.2
2
3 phases:
4   install:
5     runtime-versions:
6       java: corretto8 # Amazon Corretto 8 - production-ready distribution of the OpenJDK
7       docker: 18
8   pre_build:
9     commands:
10      - echo Region = $AWS_DEFAULT_REGION # Check Environment Variables
11      - echo Account ID = $AWS_ACCOUNT_ID # Check Environment Variables
12      - echo ECR Repo = $IMAGE_REPO_NAME # Check Environment Variables
13      - echo Docker Image Tag = $IMAGE_TAG # Check Environment Variables
14      - echo Logging in to Amazon ECR...
15      - $(aws ecr get-login --no-include-email --region $AWS_DEFAULT_REGION) # Login ECR
16   build:
17     commands:
18      - echo Build started on `date`
19      - echo Building the Docker image...
20      - mvn package -Dmaven.test.skip=true # Build maven
21      - docker build -t $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$IMAGE_TAG . # Build docker image
22   post_build:
23     commands:
24      - echo Build completed on `date`
25      - echo Pushing the Docker image...
26      - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$IMAGE_TAG # Push docker image to ECR

```

- Local 터미널에 입력하는 리눅스 명령어를 빌드 phase 별 순차적으로 정리해 놓은거라고 보면 됨
- echo 는 프린트 명령어이니 나머지 명령을 보면 됨
- CodeBuild에 입력한 환경 변수들이 이 파일에서 사용됨을 볼 수 있음(AWS_DEFAULT_REGION 등)
 - 참조 방법: **\$환경변수명**
- buildspec.yml을 포함한 프로젝트 수정사항을 github에 push
- Push 이벤트가 webhook 으로 CodeBuild를 자동 실행

msa-booking
알림
공유
편집
빌드 프로젝트

구성

소스 공급자 GitHub	기본 리포지토리 park108/msa-booking	아티팩트 업로드 위치 -	빌드 배치 비활성
------------------	---	------------------	--------------

빌드 기록
배치 이력
빌드 세부 정보
빌드 트리거
지표

빌드 기록

<input type="checkbox"/>	빌드 실행	상태	빌드 번호	소스 버전	제출자	기간
<input type="checkbox"/>	msa-booking:27b22f67-cae3-4170-a5d0-01241428fccd	진행 중	1	608522a446c91a4388fdb8c373561a30f2757c	GitHub-Hookshot/d696b2a	4초

- 빌드 실행 버전을 클릭하여 실시간 로그를 볼 수 있음

