**Food Delivery 4 Team**

* **서비스 시나리오** 
  1. **기능적 요구사항**

고객이 메뉴를 주문한다

고객이 결제한다

결제가 되면 주문 내역이 상점에 전달된다

상점 주인은 주문내역을 확인하여 메뉴를 접수하고 배송을 시작한다.

고객이 주문을 취소할 수 있다

주문이 취소되면 배송을 취소한다.

메뉴가 취소되면 결제를 취소한다.

고객이 주문상태를 중간중간 조회한다.

* 1. **비기능적 요구사항**
     1. 트랜잭션

주문이 완료되면 결제까지 한번에 처리된다. (Sync 호출)

* + 1. 장애격리

배송 기능이 수행되지 않더라도 주문, 결제는 받을 수 있어야 한다.

(Async (event-driven), Eventual Consistency)

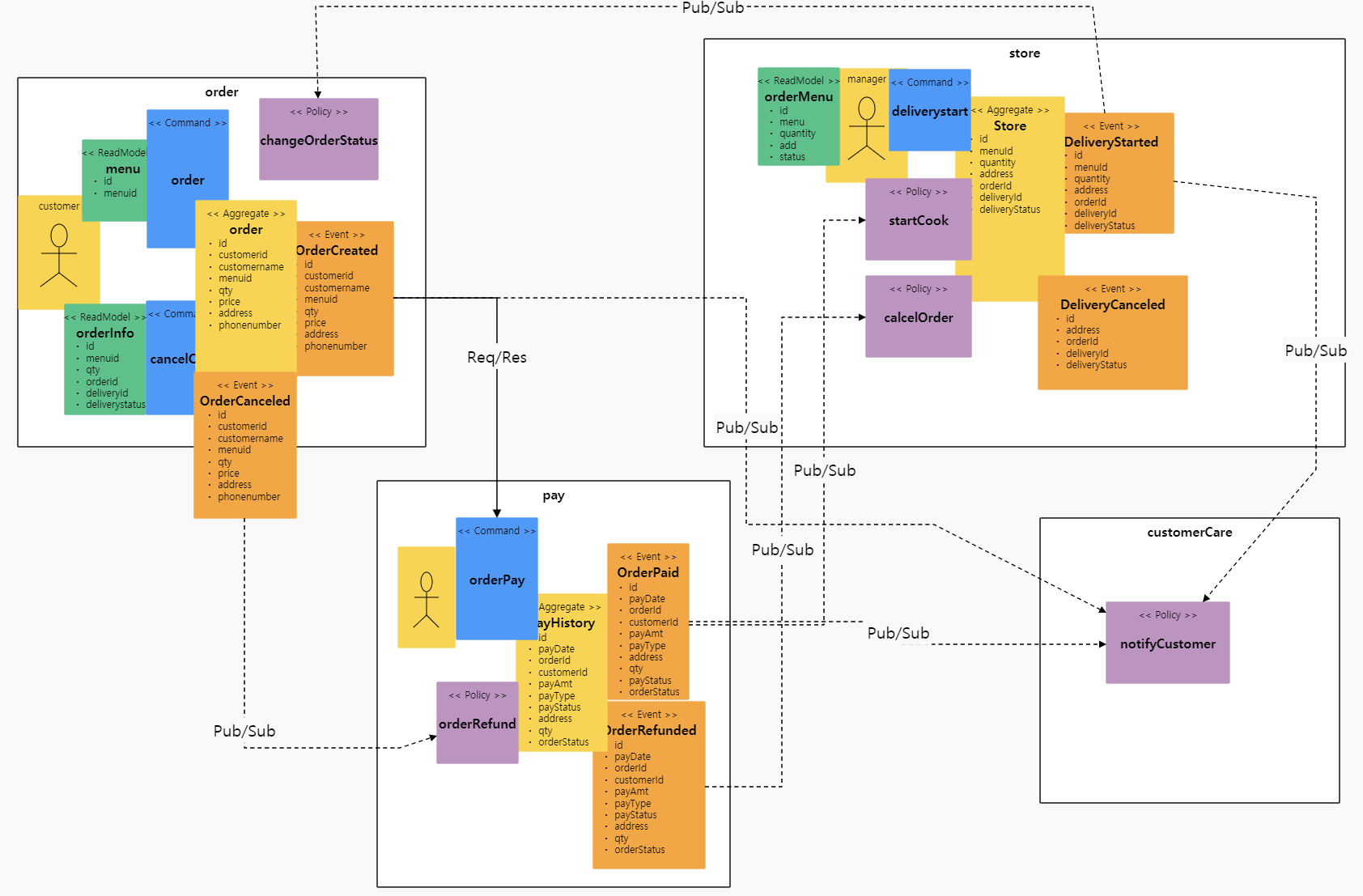
pay 시스템이 과중되면 주문을 잠시동안 받지 않고 결제를 잠시후에 하도록 유도한다. Circuit breaker, fallback

* + 1. **성능**

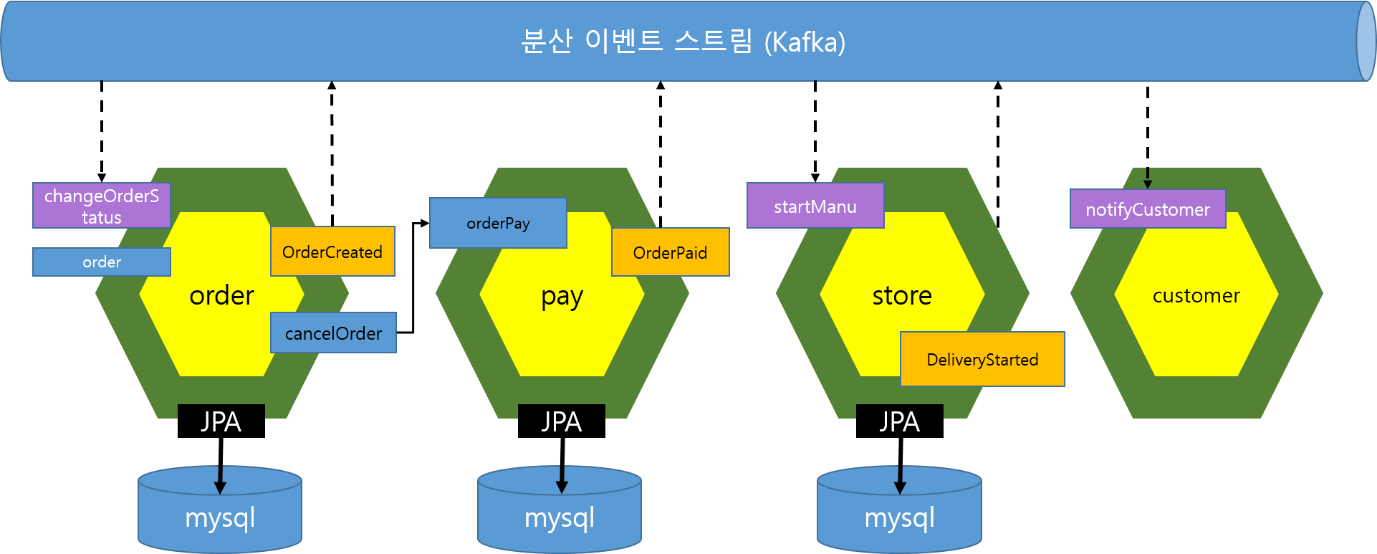
고객이 자주 확인할 수 있는 주문상태를 마이페이지(프론트엔드)에서 확인할 수 있어야 한다. CQRS

주문상태가 바뀔때마다 카톡 등으로 알림을 줄 수 있어야 한다. Event driven

* **분석 설계**



* 헥사고날 아키텍처 다이어그램 도출



* **구현**

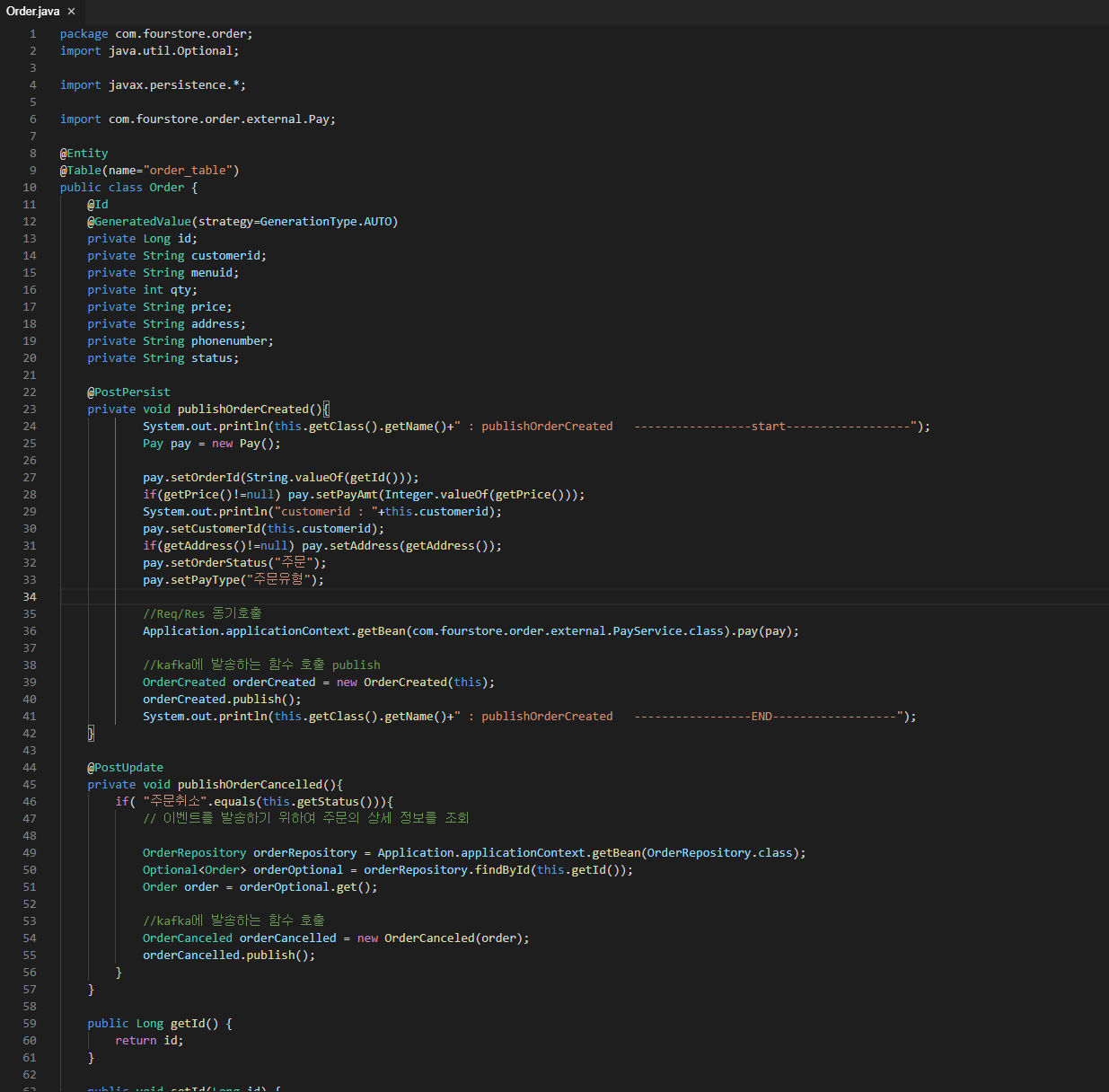
분석/설계 단계에서 도출된 헥사고날 아키텍처에 따라, 각 BC별로 대변되는 마이크로 서비스들을 스프링부트로 구현하였다.

모든 시스템은 spring-boot로 구현하였고 mvn spring-boot:run 명령어로 실행한다.

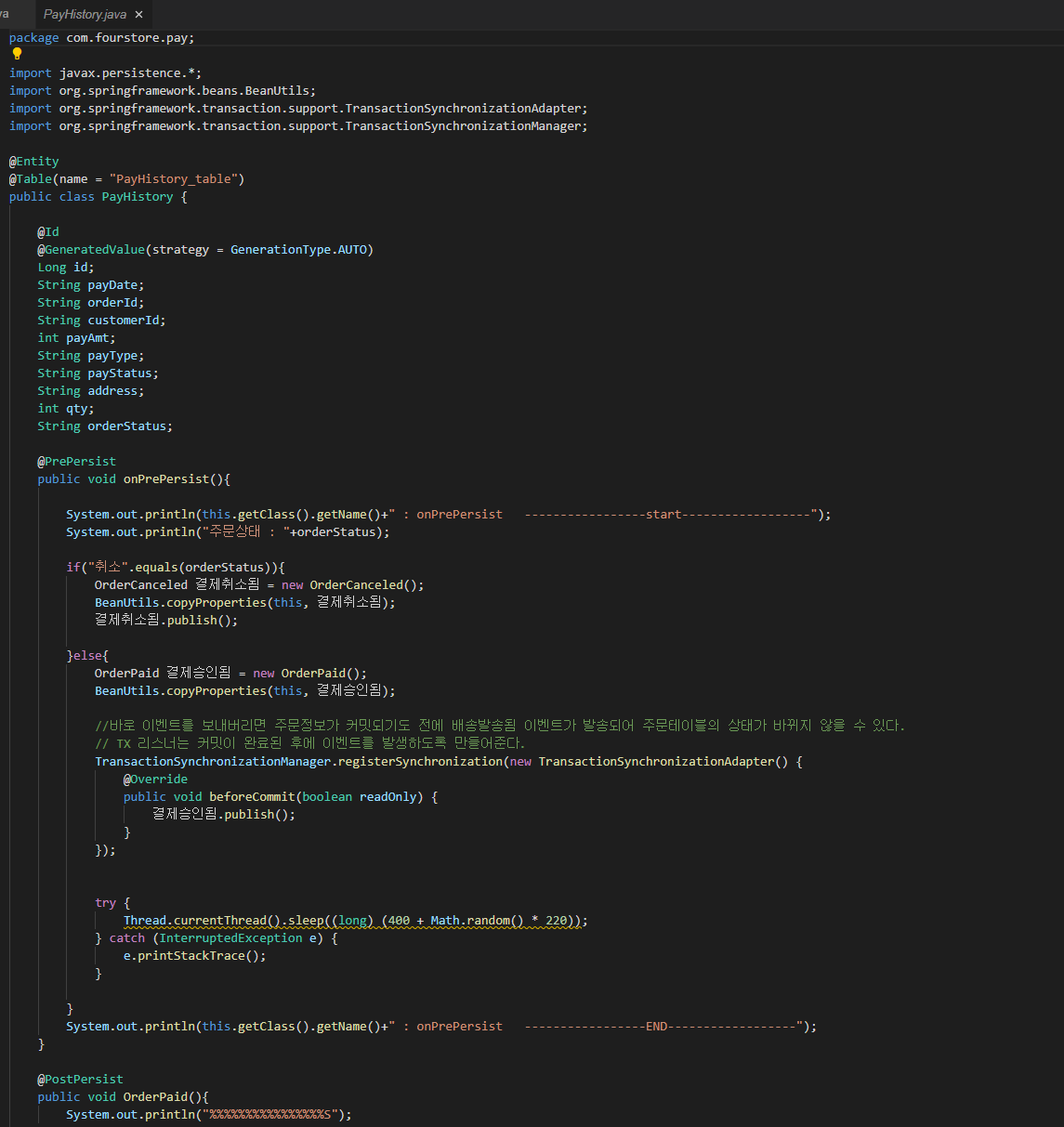
* **DDD 적용**

각 서비스내에 도출된 핵심 Aggregate Root 객체를 Entity 로 선언하고 강의시간에 예제로 든 내용 위주로 작성하였다.

Order.java

****

PayHistory.java

****

Store.java

****

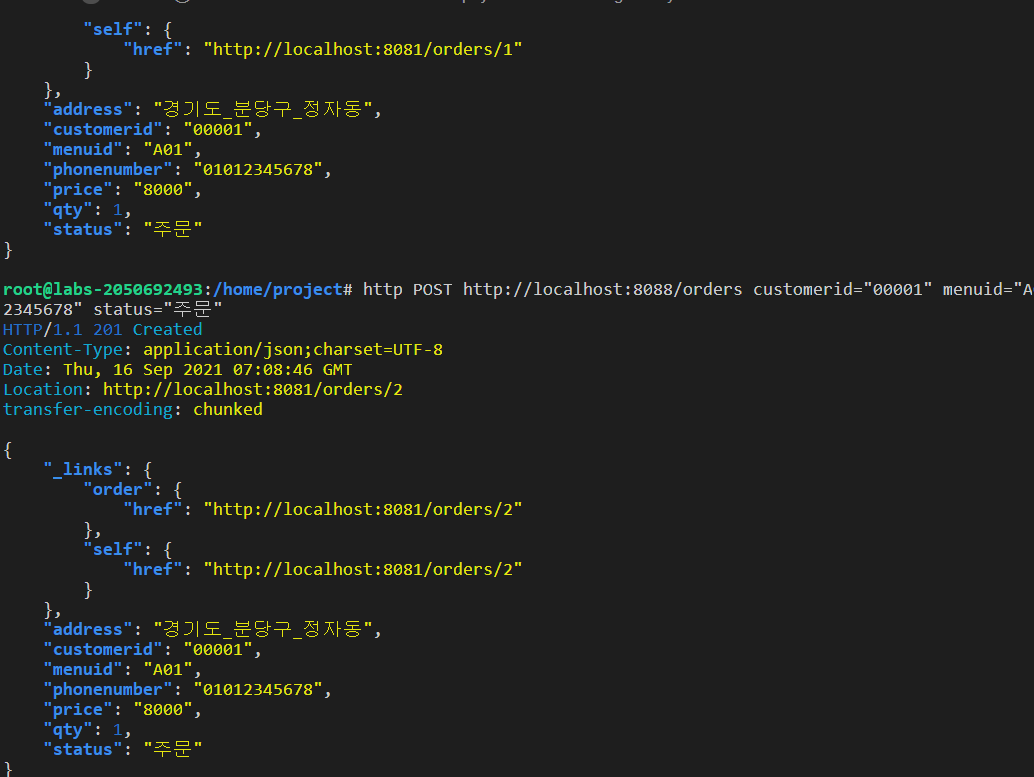
* **Saga**

- store서비스를 중단한 상태에서 여러 개의 주문처리후, store서비스를 start하였을때, 주문요청을 store에서 처리하는 것을 확인한다.

- store서비스를 중단한 상태에서 연속된 order서비스를 호출하고, 각 서비스의 데이터를 확인한다.

http POST http://localhost:8088/orders customerid="00001" menuid="A01" qty="1" price="8000" name="이름" address="경기도\_분당구\_정자동" phonenumber="01012345678" status="주문"

* 2회 실행



* pay서비스 확인: 주문내역 2건을 확인할 수 있다.



* store서비스를 실행하여 확인한다. (3건과 2건 총5건을 실행하였더니 store에도 5건)



* **CQRS**

- 주문상태 및 배송상태 등을 조회할 수 있도록 CQRS로 구현

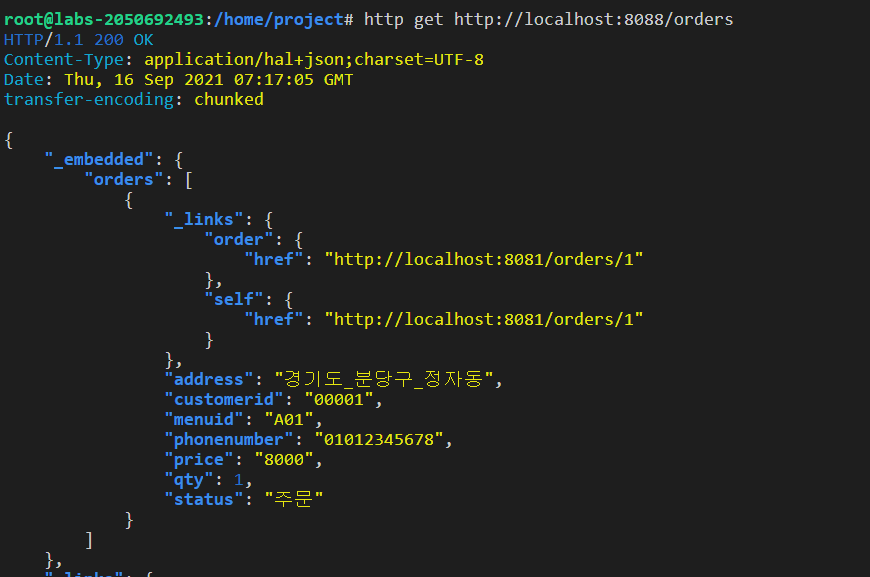
- order, pay, store의 상태를 통합해서 조회하기 때문에 타 서비스들의 성능저하 문제를 해결할 수 있다.



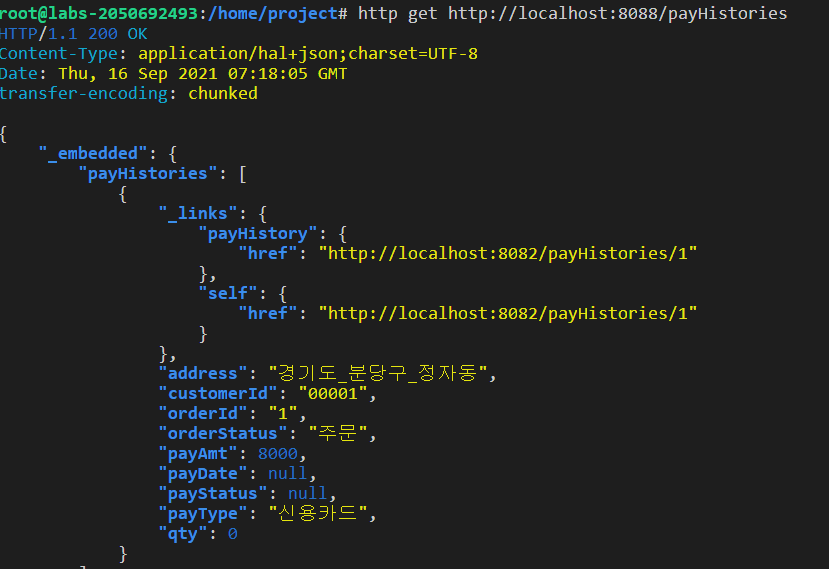
* **Correlation**

주문 후 주문취소시 order, pay의 상태를 확인한다.

Order

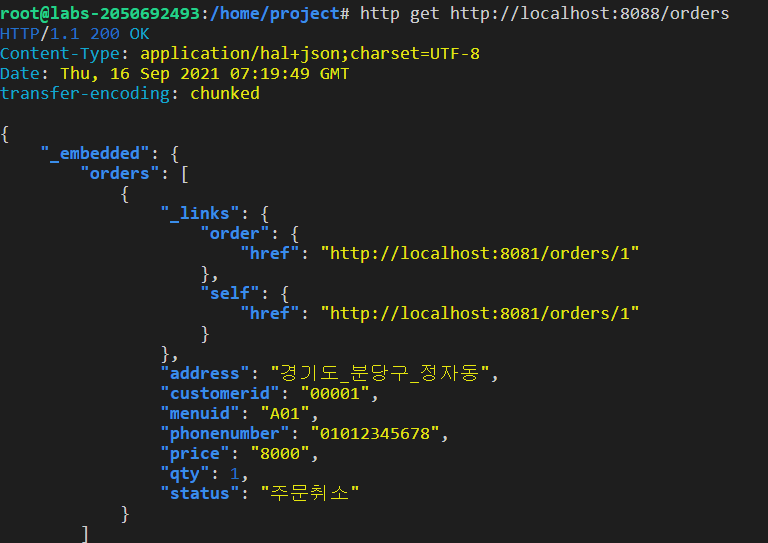


Pay

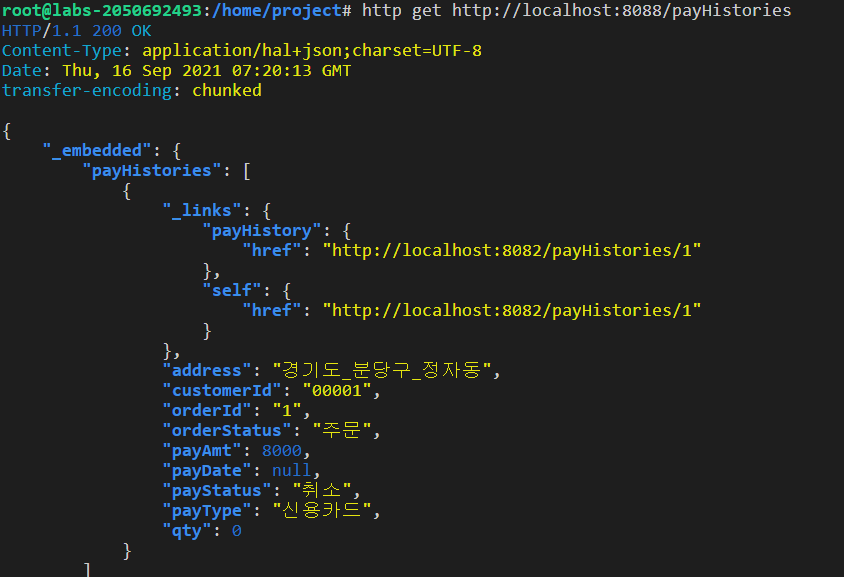


주문 후 주문취소시 order, pay의 상태를 확인한다.

Order



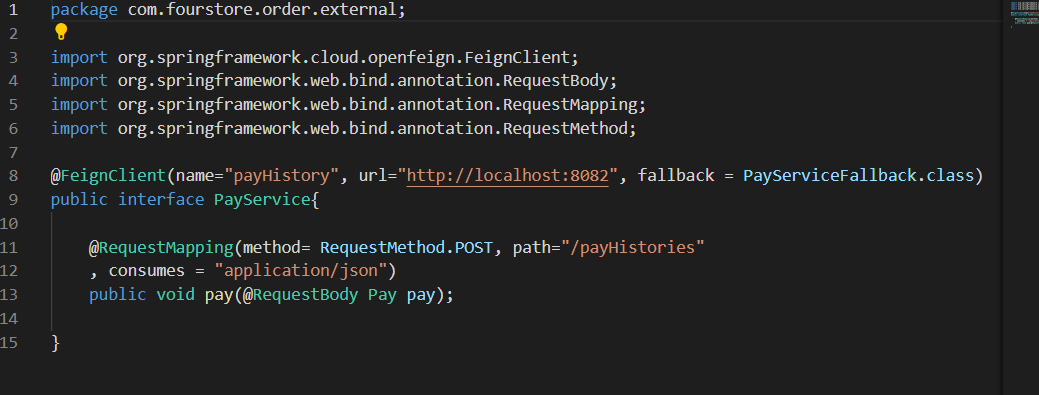
Pay



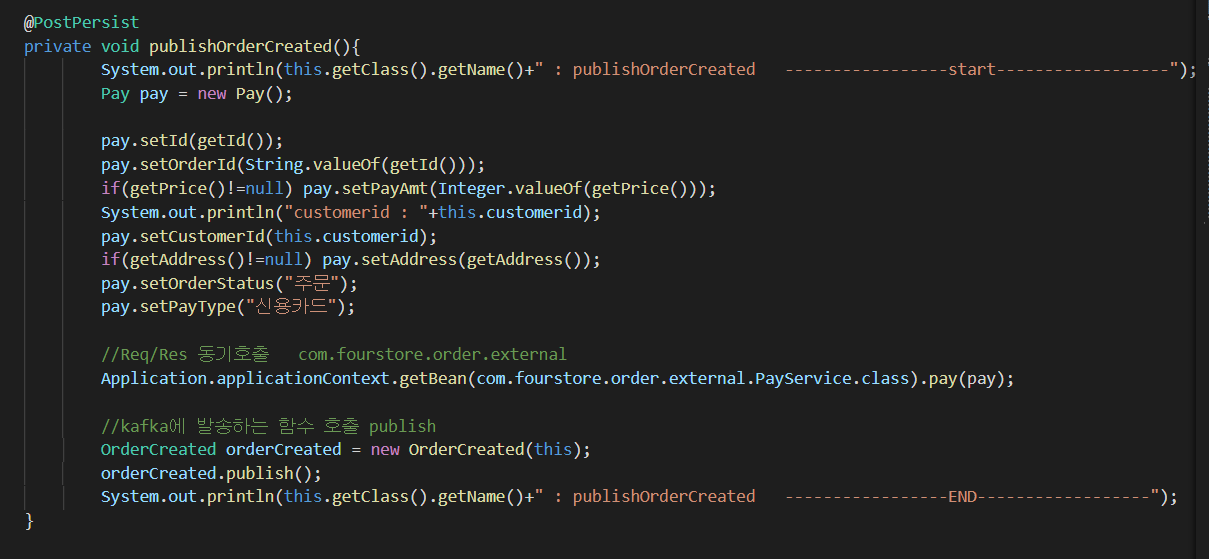
* **Req/resp**

분석단계의 중요한 요건으로 주문(order)->결제(pay)간의 호출은 동기호출로 처리하였다. 호출 프로토콜은 FeignClient를 이용하여 호출하도록 한다.

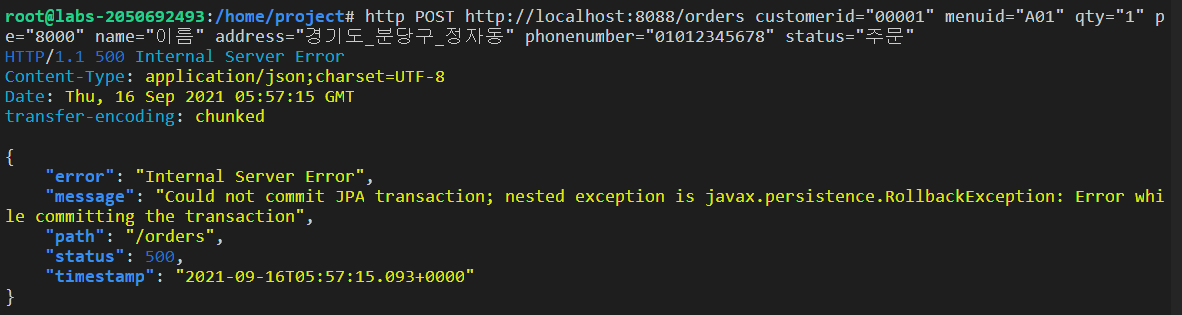
* 결제서비스를 호출하기 위하여 FeignClient를 이용하여 Service대행 인터페이스구현



* 주문완료 직후(@PostPersist) 결제를 요청



동기호출에서는 결제시스템이 장애가 나면 주문을 받지 못함을 확인 함



* **비동기식 호출**

Kafka를 이용하여 결제(pay)에서 상점(store)로 연동하는 부분을 비동기식으로 처리하였다.

Pay는 결제가 완료되면 카프카로 결제가 승인되었음을 알린다.

@PrePersist

    public void onPrePersist(){

            OrderPaid 결제승인됨 = new OrderPaid();

            BeanUtils.copyProperties(this, 결제승인됨);

            결제승인됨.publish();

            });

Store는 결제 승인 이벤트에 대해서 수신하여 처리하도록 PolicyHandler를 구현하였다.

 @StreamListener(KafkaProcessor.INPUT)

    public void startMenu(@Payload OrderPaid orderPaid){

        if(orderPaid.isMe()){

            System.out.println("##### listener : " + orderPaid.toJson());

            Store st = new Store();

            st.setOrderId(Long.valueOf(orderPaid.getOrderId()));

            st.setAddress(orderPaid.address);

            st.setDeliveryId((long)0);

            st.setDeliveryStatus("배송");

            st.setOrderId(Long.valueOf(orderPaid.getOrderId()));

            st.setQuantity(orderPaid.getQty());

            storeRepository.save(st);

        }

    }

Store는 order/pay와 분리되어 있으며, 이벤트 수신에 따라 처리되기 때문에, store가 유지보수로 인해 잠시 내려간 상태라도 주문을 받는데 문제가 없다.

#store를 정지시킴

#주문처리

http localhost:8081/orders menuid=coke #Success

#주문상태 확인

http loacalhost:8080/orders #주문상태 안바뀜

"address": null,

"customerid": null,

"menuid": "coke",

"phonenumber": null,

"price": null,

"qty": 0,

"status": “결제완료”

#상점 서비스 기동

#주문상태 확인

http localhost:8080/orders #주문들이 배송처리됨

"address": null,

"customerid": null,

"menuid": "coke",

"phonenumber": null,

"price": null,

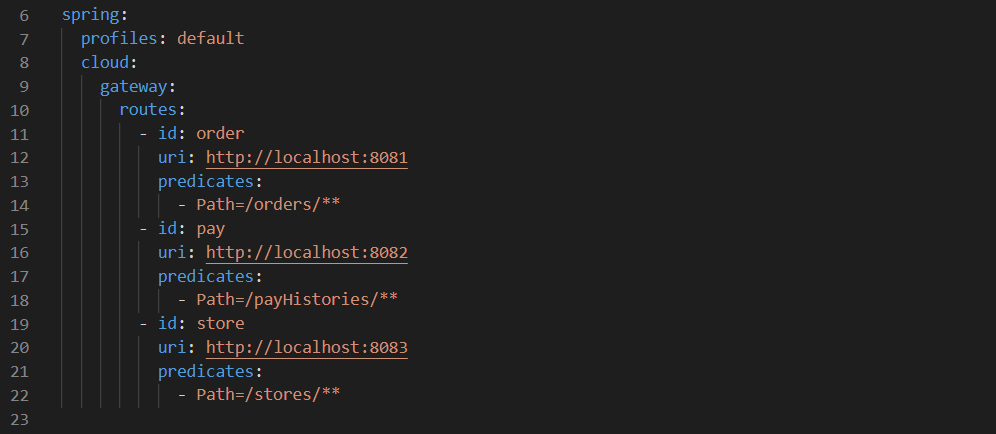
"qty": 0,

"status": “배송출발”

* **API gateway**

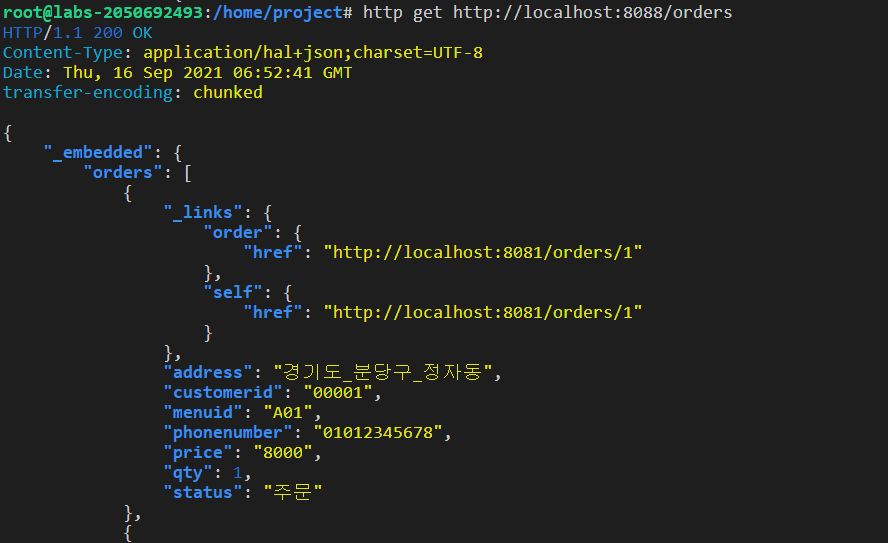
Local 테스트 환경에서는 localhost:8088에서 Gateway API 가 작동.

application.yml 파일에 프로파일 별로 Gateway 설정.

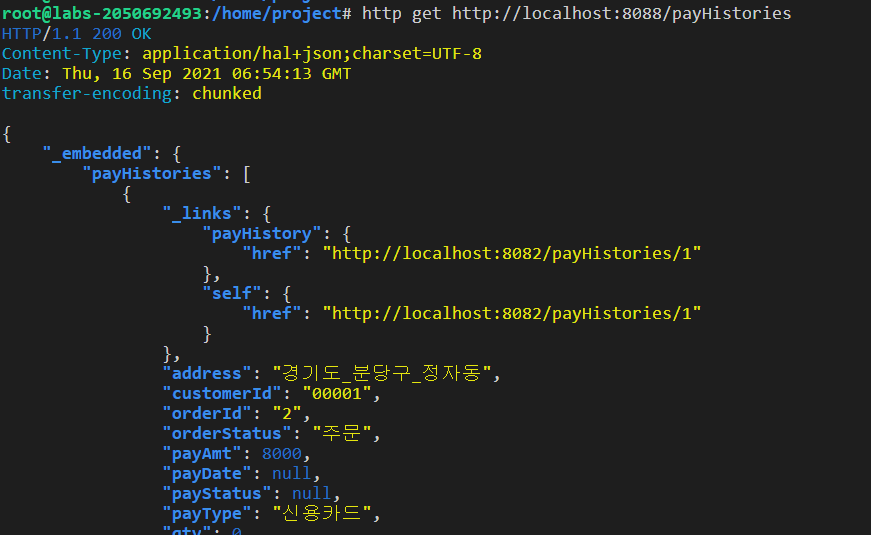


* GateWay 서비스 기동 후, GateWay 서비스의 port 8088로 order, pay, store 서비스가 접속가능한지 확인

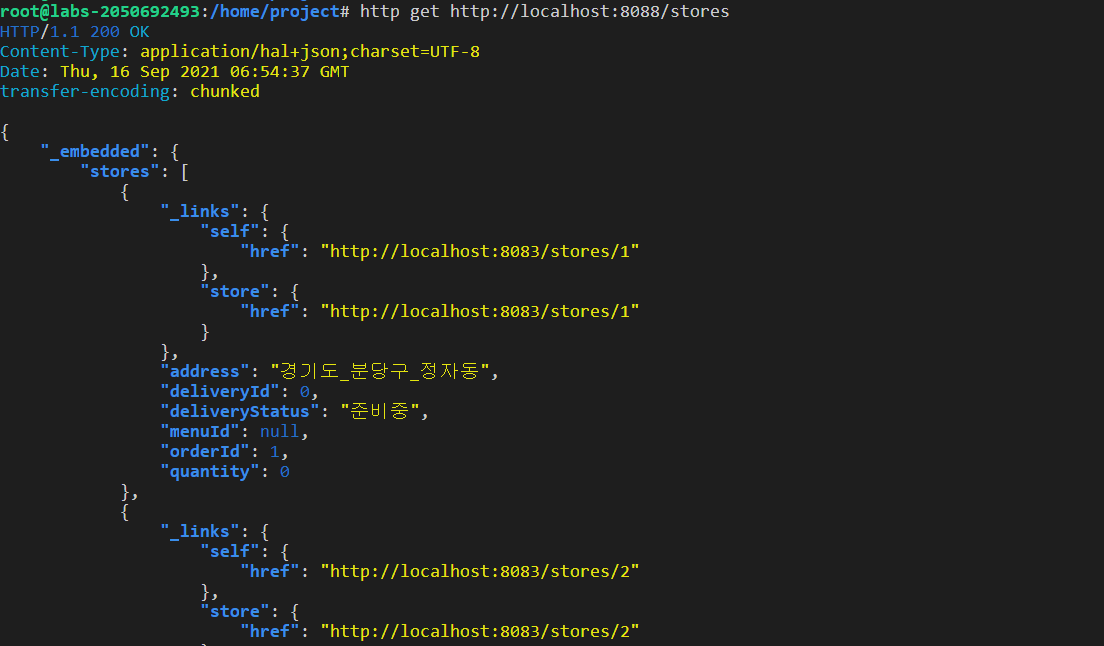
http get <http://localhost:8088/orders>



http get [http://localhost:8088/payHistories](http://localhost:8088/orders)

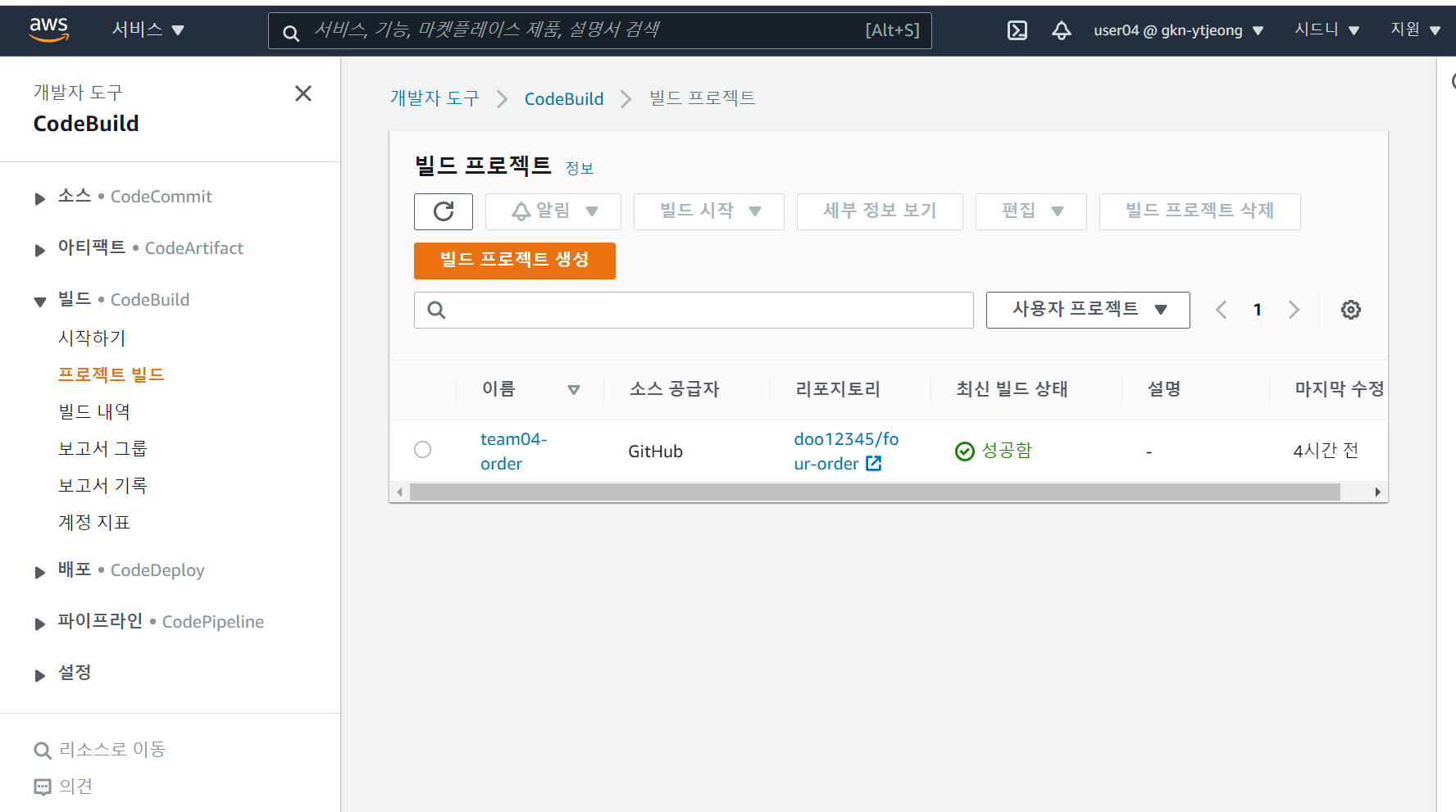


http get <http://localhost:8088/stores>



* **CI/CD 설정**

Order에 적용하였으며, AWS Codebuild를 사용하여 적용했고, pipeline build script는 order 프로젝트 하위 cloudbuild.yaml에 포함되어 있다.



* **동기식 호출/서킷브레이킹/장애격리**

Spring FeignClient + Hystrix 옵션을 사용하여 구현함.

주문(order)🡪결제(pay) 시의 연결을 RESTful Req/Res로 연동하여 구현, 결제 요청이 과도할 경우 서킷브레이커를 통하여 장애격리 하였음.

Hystrix 설정: 요청 쓰레드에 처리시간 610 밀리가 넘어 어느정도 유지되면 회로 차단 설정

pay 내 application.yml 파일에 설정

feign:

  hystrix:

    enabled: true

hystrix:

  command:

    # 전역설정

    default:

      execution.isolation.thread.timeoutInMilliseconds: 610

결제(pay)에 임의 부하처리: -400 밀리에서 증감 220 밀리

payHistory.java (Entity)

@PrePersist

    public void onPrePersist(){

        ...

            try {

                Thread.currentThread().sleep((long) (400 + Math.random() \* 220));

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

부하테스터를 사용해 서킷브레이커 동작 확인:

동시사용자 100명 / 60초 실시

siege -c100 -t60s -r10 --content-type "application/json" 'http://localhost:8081/orders POST {"itemid":"coffee"}'

Transactions: 1329 hits

Availability: 62.38 %

Elapsed time: 59.82 secs

Data transferred: 0.39 MB

Response time: 4.35 secs

Transaction rate: 22.22 trans/sec

Throughput: 0.01 MB/sec

Concurrency: 96.63

Successful transactions: 829

Failed transactions: 500

Longest transaction: 8.45

Shortest transaction: 0.00

시스템이 지속적으로 CB에 의하여 자원을 보호하고 있음을 보여주나 62.38%만 결제 성공.

* **오토스케일 아웃**

결제서비스에 대한 replica를 동적으로 늘려주도록 HPA를 설정. CPU 사용량이 15프로를 넘어서면 replica를 10개까지 늘려준다.

Kubectl autoscale deploy pay --min=1 --max=10 --cpu-percent=15

Siege를 이용하여 부하를 걸어주고 모니터링을 한다.

Kubectl get deploy pay –w

시간이 지나 후 스케일 아웃이 벌어진 걸 확인

NAME READY UP-TO-DATE AVAILABLE AGE

pay 1/1 1 1 60m

pay 1/1 1 1 15m

pay 1/1 1 1 3m

siege 로그로 성공률이 높아 진 걸 확인

Transactions: 1373 hits

Availability: 93.28 %

* **무정지 재배포**

Seige로 배포작업 모니터링

siege -c100 –t120s -r10 --content-type "application/json" 'http://localhost:8081/orders POST {"itemid":"coffee"}'

새버전 배포 시작 후 siege 확인

Transactions: 2704 hits

Availability: 73.96 %

배포 중 성공률이 떨어지는 걸 확인. deployment.yml에 readiness probe 설정

readinessProbe:

            httpGet:

              path: '/actuator/health'

              port: 8080

            initialDelaySeconds: 10

            timeoutSeconds: 2

            periodSeconds: 5

            failureThreshold: 10

설정 후 배포 값

Transactions: 2704 hits

Availability: 100.00 %

무정지 배포 확인