커피 전문점 MSA 시스템

Spring Cloud & Kubernetes 기반의
확장 가능한 마이크로서비스 아키텍처 설계 및 구현 전략

Java 17 | Spring Boot 3.1 | Spring Cloud | Kubernetes | Rabbit MQ

Presented By: 팀 부리또

Project Overview

프로젝트 소개: 대규모 프랜차이즈를 위한 통합 주문·재고 관리 플랫폼

Why: 대형 커피 프랜차이즈는 **지점별 주문 트래픽 편차**, **실시간 재고 불일치**, **운영 기능의 분산**으로 점주와 본사 모두 운영 효율에 한계를 겪고 있다.

What: 본 프로젝트는 **주문·재고·운영 기능을 하나의 플랫폼으로 통합**하여 지점 단위 운영 안정성과 관리 효율을 동시에 확보하는 것을 목표로 한다.

Project basic info

프로젝트 팀 구성 및 역할

훈련생	역할	담당 업무
서재홍	팀장	Frontend Service 개발, 시스템 아키텍처 설계, Docker/k8s 배포
하성호	팀원	Order service, Inventory Service 개발, RabbitMQ 설계, REST API 구현, ERD 설계, 데이터 모델링
박윤호	팀원	Product Service, Inventory Service 개발, REST API 구현, ERD 설계, 데이터 모델링
이주희	팀원	Member Service, Board Service, Admin Service 개발 REST API구현, ERD 설계, 데이터 모델링

Project basic info

프로젝트 수행 절차 및 방법

작 업 명		담당자	개발 일정			비고
			1 Week	2 Week	3 Week	
계획	목표 정의	전체				2일 소요
	계획서 작성	전체				
	요구사항 분석	전체				
	계획서 작성	전체				
설계	시스템 아키텍처	전체				4일 소요
	ERD 설계	전체				
	화면 설계	서재홍				
구현	기능 구현	전체				
통합 및 테스트	서비스 통합	박윤호/ 서재홍				9일 소요
	RabbitMQ 연동	박윤호				
	단위/통합 테스트	전체				
	오류 수정	전체				
배포 및 발표	Docker 컨테이너화	서재홍				2일 소요
	K8s(EKS) 배포	서재홍				

Technical Foundation

기술 스택 (Tech Stack)

LANGUAGE

JDK 17 LTS

Java

Core Language

Record, Sealed Class 등 최신 문법 활용 및 장기 지원 버전 안정성 확보

FRAMEWORK

v3.1.5

Spring Boot

App Framework

Spring 6 기반, 빠른 구동 및 효율적인 마이크로서비스 설정 관리

CLOUD NATIVE

2022.0.4

Spring Cloud

MSA Toolkit

Gateway, Eureka, OpenFeign 등 분산 시스템 패턴 구현

INFRASTRUCTURE

AWS EKS

Kubernetes

Orchestration

컨테이너 자동 배포, 스케일링, 복구 및 서비스 디스커버리 연동

MESSAGING

3.x

RabbitMQ

Event Bus

서비스 간 비동기 이벤트 처리 (주문→재고) 및 결합도 감소

DATABASE

8.0

MySQL

RDBMS

서비스별 Database 분리 (Polyglot Persistence 준비) 및 트랜잭션 관리

SECURITY

JJWT 0.11.5

JWT & Security

Authentication

Stateless 인증 아키텍처, Gateway 레벨의 토큰 검증 및 라우팅 제어

DEVOPS

Gradle 7+

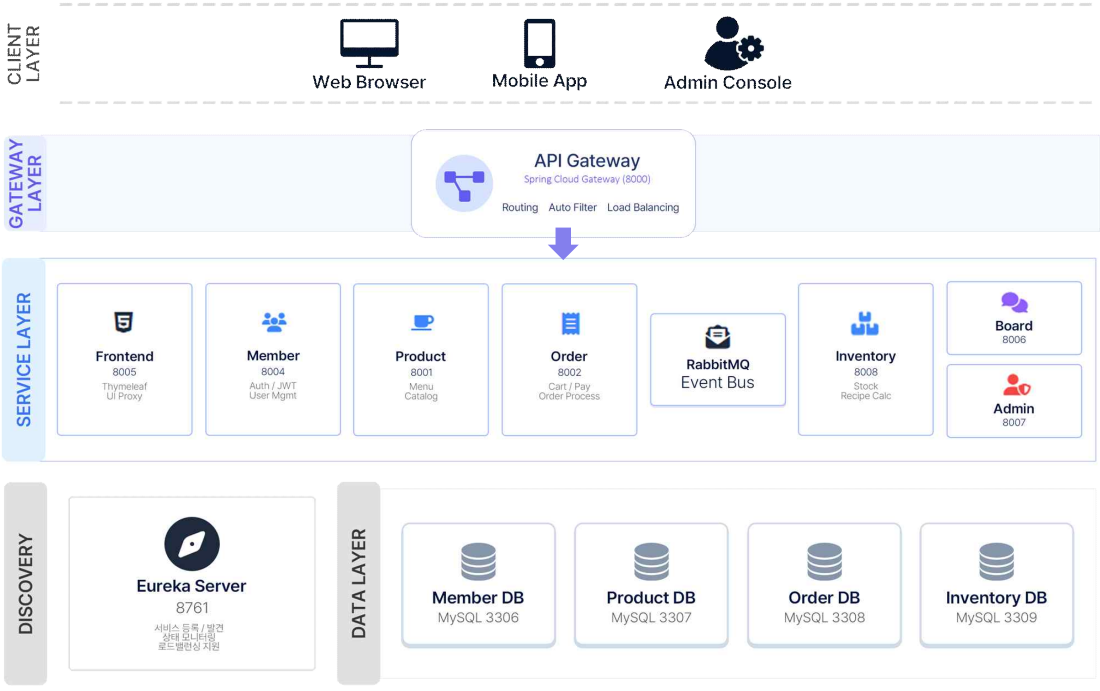
Docker

Containerization

멀티 모듈 빌드 구성 및 경량 컨테이너 이미지 최적화

ARCHITECTURE Micro Services Overview

시스템 아키텍처 구성도



STRATEGY Domain-Driven Design

서비스 분리 전략

분리 원칙

Key principles

Bounded Context

비즈니스 도메인 경계에 따라 서비스를 정의하여 역할과 책임을 명확히 구분

Loose Coupling

서비스 간 직접 참조를 제거하고 API 및 메시지를 통해서만 상호작용

DB per Service

각 서비스는 자체 데이터베이스를 소유하며 타 서비스 데이터 직접 접근 차단

총 서비스 수 **7 Services**

CORE BUSINESS DOMAIN

Product Service
상품 관리

- 메뉴 카테고리 및 상세 정보관리
- 옵션(사이즈, 시럽 등) 구성
- 레시피 및 영양 정보 매핑

Order Service
주문 처리

- 장바구니 관리 및 주문 생성
- 주문 상태 관리 (접수 / 완료 / 취소)
- 결제 승인 및 이력 관리
- **주문 완료 이벤트 발행**

Inventory Service
재고 관리

- 원자재 마스터 관리
- 실시간 재고 수량 추적
- 레시피 기반 재고 차감 계산
- **주문 이벤트 구독 처리**

SUPPORTING & INTERFACE

Member Service
회원 / 인증

- 사용자 가입 및 정보 관리
- **JWT 토큰 발급 및 검증**
- 로그인 실패 보안 정책

Board Service
게시판

- 공지사항 및 FAQ 관리
- 1 : 1 문의 및 답변 처리
- 댓글 작성 기능

Interface Layer
Admin & Frontend

Admin (8007)
- 운영자 백오피스
- 통계 대시보드

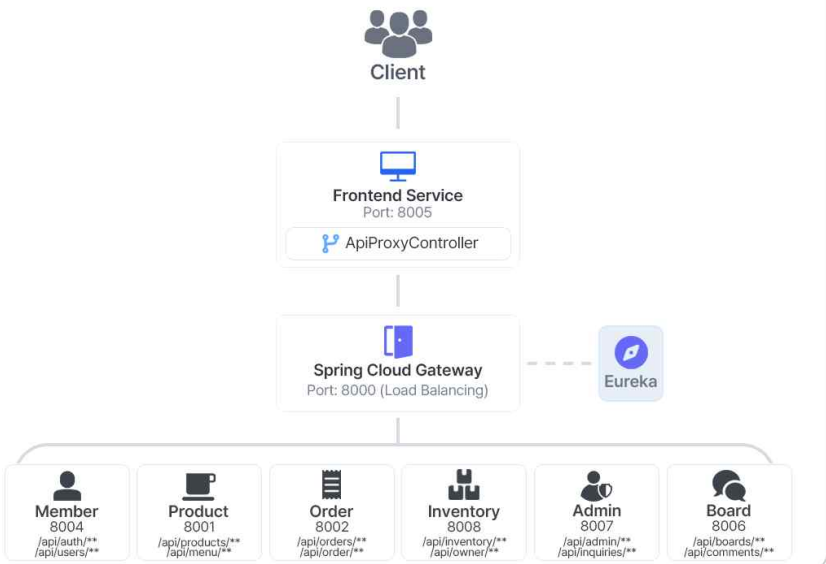
Frontend (8005)
- Thymeleaf UI
- 사용자 접점

STRATEGY Domain-Driven Design

서비스 간 통신 전략

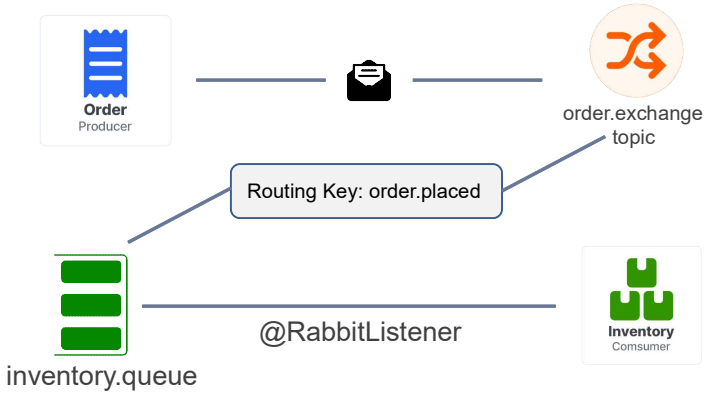
동기 통신: REST API Flow

즉시 응답 필요



비동기 통신: RabbitMQ

후속 처리



설계 이유

- 사용자 경험을 위한 즉시 응답 (메뉴 조회 등)
- 트랜잭션 분리가 가능한 후속 작업 비동기화

아키텍처 이점

- 결합도 감소: 서비스 간 직접 호출 제거
- 장애 격리: 재고 서비스 장애 시 주문 접수 가능

DATA LAYER Database per Service Pattern

데이터베이스 독립화

물리적 분리 구조

member_db 3306
Service: Member

Tables
- users
- inquiries
- comments
- notices
- faqs

product_db 3307
Service: Product

Tables
- menu
- allergy
- nutrition
- recipe
- menu_option
- option_master
- material_master

order_db 3308
Service: Order

Tables
- orders
- order_option
- order_item
- cart_header
- cart_option
- cart_item

inventory_db 3308
Service: Inventory

Tables
- material_master
- option_master
- recipe

설계 전략 및 패턴

장애 격리 (Isolation)

한 서비스의 DB 장애가 전체 시스템으로 전파되는 것을 방지합니다. 특정 DB(예
가 다운되더라도 주문 접수 등의 핵심 기능은 유지될 수 있도록 설계되었습니다

참조 무결성 (No Foreign Keys)

서비스 간 Foreign Key 제약 조건을 제거하고, 논리적인 ID(문자열/Long)로만 참조합니다.

~~JOIN users u ON o.user_id = u.id~~ → API Call / ID Reference

비정규화 (De-normalization)

데이터 변경(가격 인상 등)에 영향을 받지 않도록 주문 생성 시점에 핵심 데이터(상품명, 단가)를 복제하여 저장합니다.

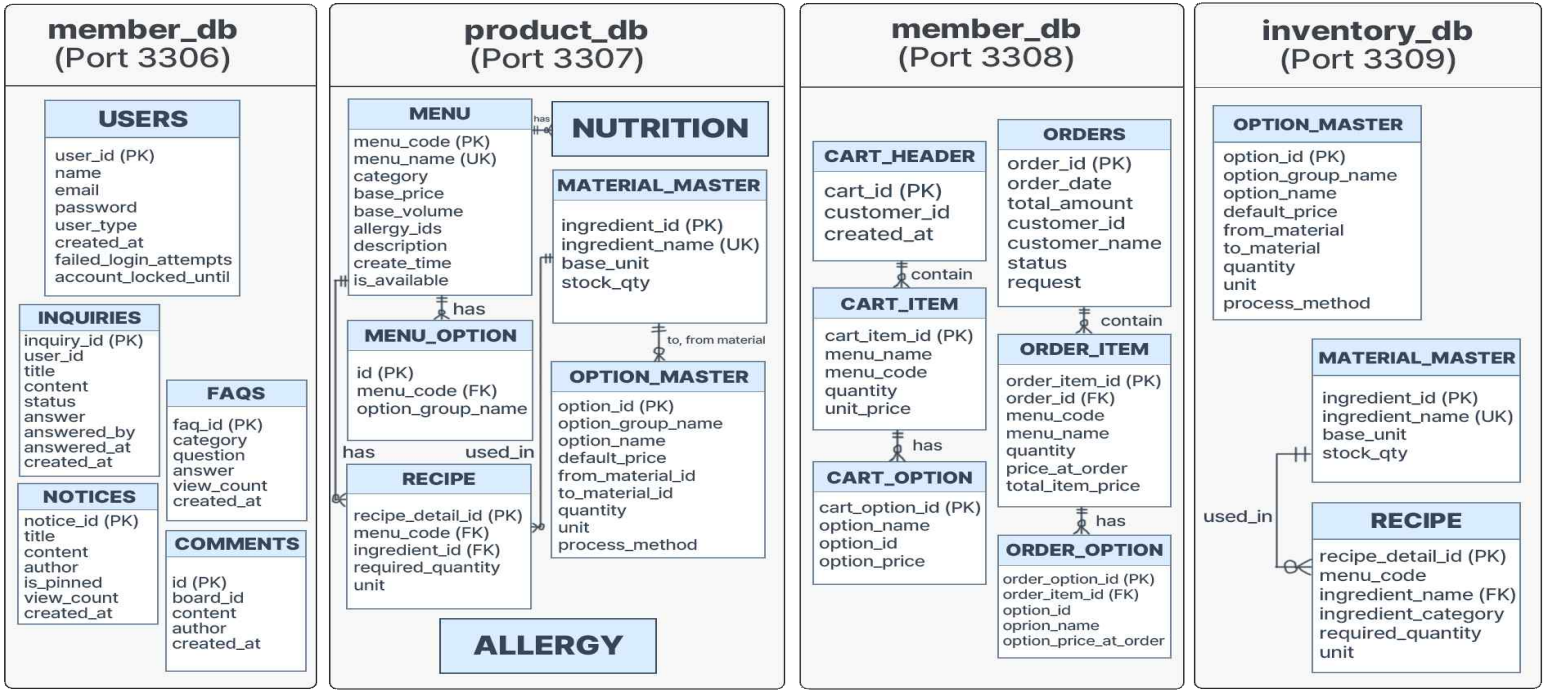
Product DB
가격: 5,000원

→ Copy at Order Time

Order DB
결제 금액: 5,000원
(가격 변동 무관)

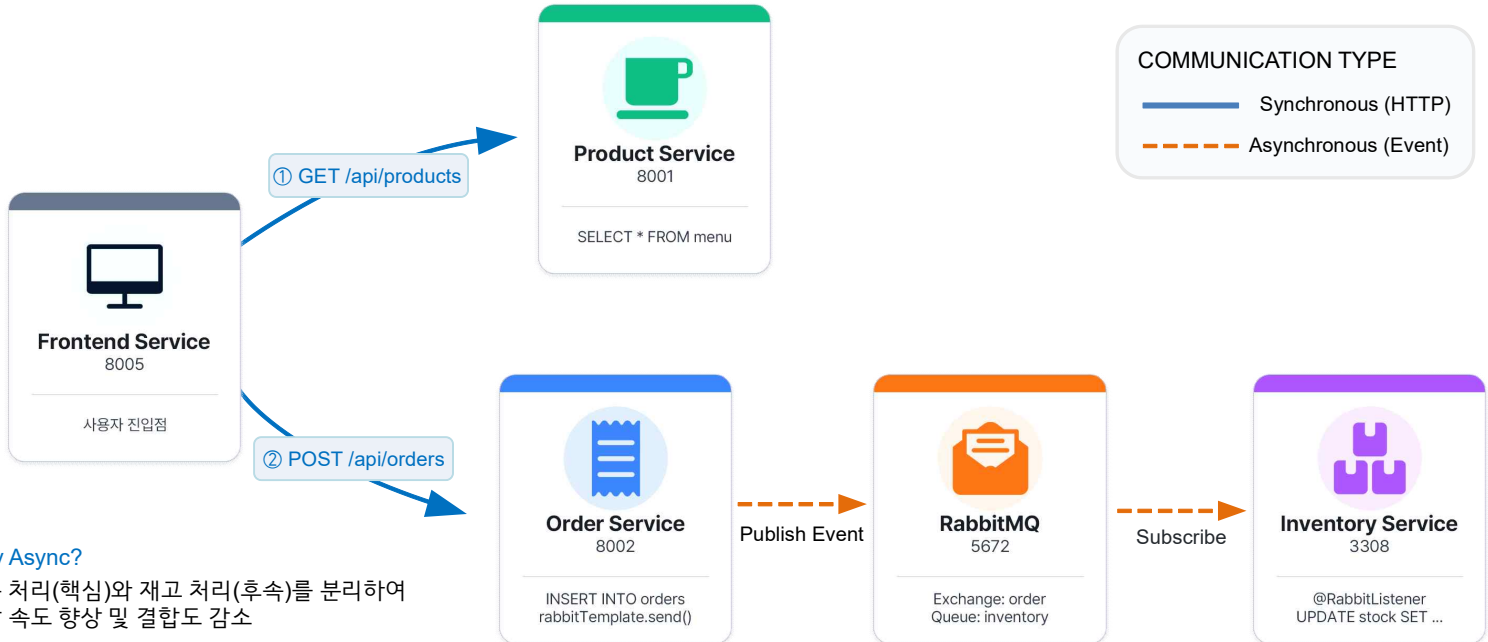
DATABASE Database ERD

데이터베이스 ERD



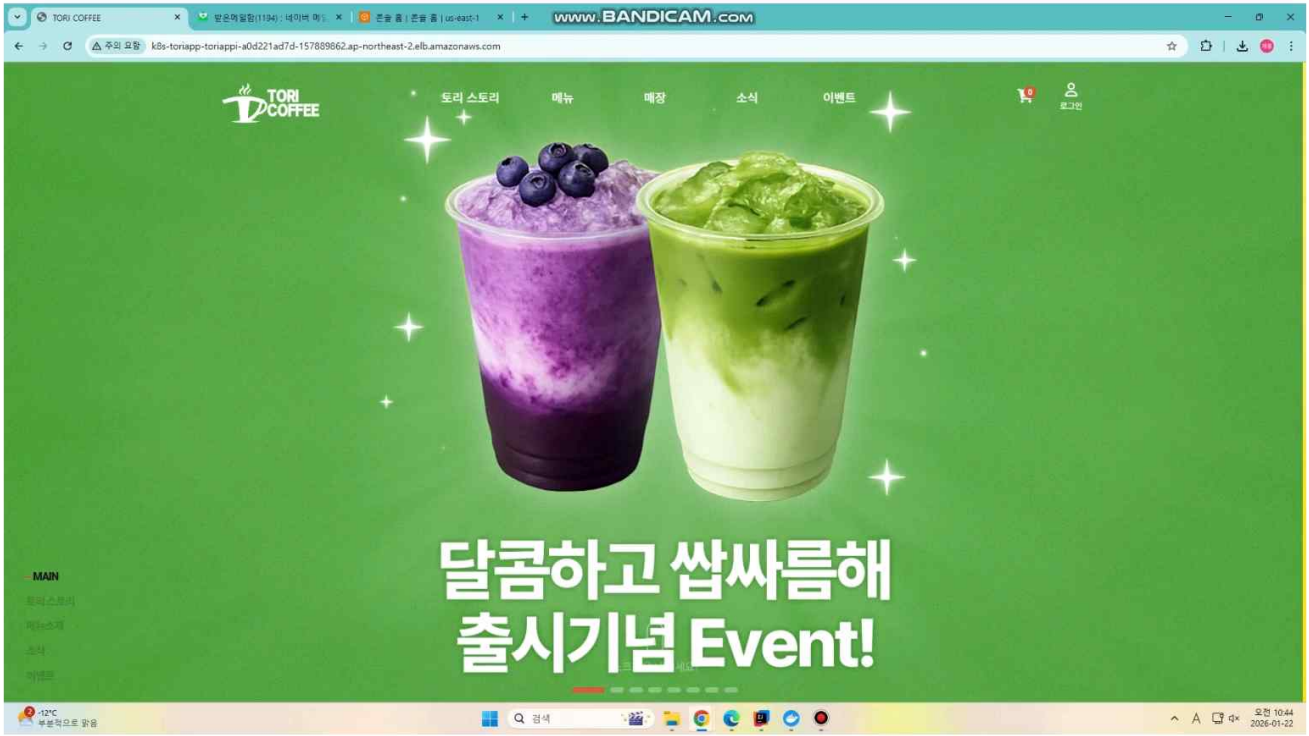
PROCESS FLOW Order Processing Pipeline

주문 처리 플로우



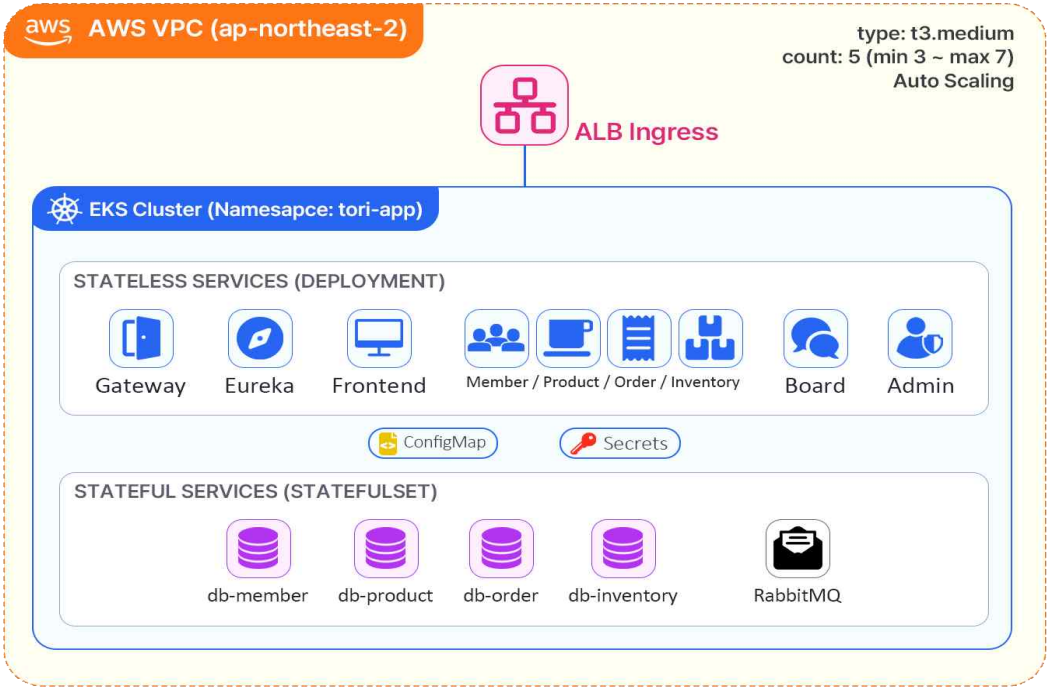
Why Async?

주문 처리(핵심)와 재고 처리(후속)를 분리하여
응답 속도 향상 및 결합도 감소


TORI COFFEE
토리 스토리
메뉴
매장
소식
이벤트
로그인
달콤하고 쌉싸름해
출시기념 Event!
MAIN

DEPLOYMENT AWS EKS Architecture

Kubernetes 인프라 및 배포 환경



Deployment (Stateless)

Spring Boot 서비스에 적용
Pod가 자유롭게 생성/삭제되며,
HPA를 통해 트래픽에 따라 자동 확장됨

```
kind: Deployment replicas: 2 resources: requests: { cpu: 250m, mem: 256Mi } limits: { cpu: 500m, mem: 512Mi }
```

StatefulSet (Stateful)

MySQL, RabbitMQ 등 데이터 영속성이 필요한 서비스
고정된 호스트명(ordinal index)과
PVC(영구 볼륨)를 보장

```
kind: StatefulSet serviceName: "mysql-member" volumeClaimTemplates: - metadata: { name: data } spec: { accessModes: [ "ReadWriteOnce" ] }
```

Ingress & Config

- **ALB Ingress:** /api/* 경로 기반 라우팅
- **ConfigMap:** 환경 변수 (DB host, RabbitMQ URL)
- **Secrets:** DB Password, 인증 키 (Base 64)

CHALLENGES Architecture Decisions & Trade-offs

기술적 도전과 해결

PROBLEM 01

트래픽 편차로 인한 서비스 장애 위험

대형 프랜차이즈 특성상 특정 시간대·지점에 주문 트래픽이 집중되면서 단일 서비스 구조에서는 장애 전파 및 확장 한계가 발생.

SOLUTION

MSA 기반 서비스 격리

MSA 기반으로 주문, 재고, 회원, 운영 기능을 분리하고, 각 서비스별 독립 DB 및 배포 구조를 설계하여 트래픽 집중 시에도 부분 확장 및 장애 격리가 가능하도록 구현.

PROBLEM 02

유지 보수 및 배포 리스크

서비스 간 직접 호출 구조는 하나의 변경이 전체 시스템 배포로 이어져 운영 안정성과 개발 생산성을 저해.

SOLUTION

라우팅·부하 분산 중앙화

Spring Cloud Eureka 기반 서비스 디스커버리와 API Gateway를 도입하여 동적 서비스 등록 및 라우팅·부하 분산을 중앙화함으로써 서비스 간 결합도를 최소화하고 독립적 배포 환경을 구축.

PROBLEM 03

실시간 정합성 유지의 어려움

주문 발생 시 재고를 동기 방식으로 처리할 경우 응답 지연 및 장애 전파 위험이 발생하였으며, 트래픽 증가 시 시스템 병목 현상이 우려.

SOLUTION

이벤트 중심 비동기 아키텍처

RabbitMQ 기반 이벤트 중심 비동기 아키텍처를 적용하여 주문 서비스는 이벤트 발행에만 집중, 재고 서비스는 이를 구독하여 재고를 실시간 반영하도록 설계함으로써 성능 저하 없이 데이터 정합성을 유지.

PROBLEM 04

트래픽 증가에 대한 인프라 대응 한계

프로모션·이벤트 기간에 급격한 트래픽 증가가 발생할 경우 수동 확장 방식은 대응 속도와 운영 안정성에 한계 존재.

SOLUTION

자동 스케일링 정책

AWS EKS 환경에서 HPA(Horizontal Pod Autoscaler) 를 적용하여 CPU 사용률 기반 자동 확장 구조를 구축하고, 트래픽 변화에 따라 서비스 인스턴스가 자동으로 증감되도록 구성.

OUTCOMES Achievements & Lessons Learned

성과 및 배운 점

프로젝트 성과

Key Achievements

아키텍처 관점에서의 설계 역량 향상 — 서재홍
MVC 패턴을 직접 구현하며 데이터 처리, 화면 구성, 비즈니스 로직을 명확히 분리한 구조를 설계하였고, 기능 확장 및 유지보수에 유리한 코드 구조를 구현하였다.

협업 중심 개발 프로세스에 대한 이해 — 하성호
MSA 구조를 기반으로 서비스 단위 역할을 분리하고, API 명세를 사전에 정의하여 병렬 개발을 진행함으로써 코드 충돌을 최소화하고 개발 효율을 향상시켰다.

분산 환경에서의 데이터 신뢰성 확보 경험 — 박윤호
서비스별 독립 DB 환경에서 발생할 수 있는데이터 정합성 문제를 고려하여 RabbitMQ 기반 비동기 이벤트 구조를 설계하고 적용하였다.

기술적 교훈

Key Learnings

구조 분리가 유지보수성과 확장성을 결정 — 이주희
MVC 패턴은 단순한 이론이 아니라, 변경에 강한 구조를 만들기 위한 설계 원칙임을 실습을 통해 체득.

협업의 핵심은 인터페이스와 책임 분리 — 박윤호
협업에서 가장 중요한 것은 구현 속도가 아니라 명확한 인터페이스 정의와 책임 분리임을 확인.

안정적인 데이터 흐름이 중요 — 하성호
분산 시스템에서는 완전한 동기화보다장애에 강한 데이터 흐름과 정합성 전략이 더 중요하다는 것을 이해하였다.

다음 단계

Future Roadmap

코드 구조 개선을 통한 설계 완성도 향상
서비스와 컨트롤러 역할을 더 명확히 분리하며, 코드 구조를 점진적으로 개선.

API 문서화 기반 협업 프로세스 정리
Swagger를 활용해 API 문서를 정리하고, 협업 흐름을 더 명확히 관리.

예외 상황을 고려한 데이터 처리 로직 보완
주문·재고 흐름에서 발생할 수 있는 예외 상황을 더 정리해 처리 로직을 보완.

Q&A

경청해 주셔서
감사합니다.