API Composition과 CQRS

도인한

API Composition

API Composition 최대한 사용하기

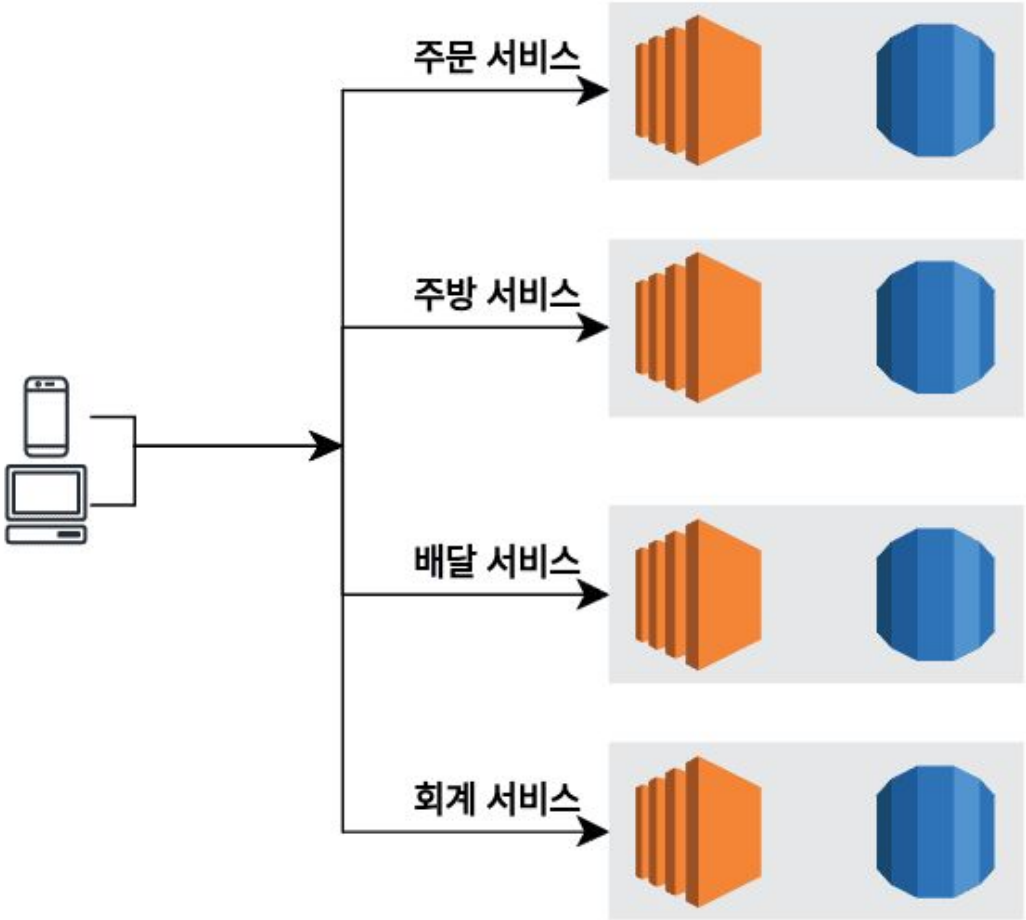
직관적이면서 조직의 개발 생산성에 도움됨

Asynchronous & Non-Blocking I/O

API Composition 역할에 필수적인 아키텍쳐

장/단점

도입 전, MSA로부터 클라이언트가 고통받는 구조



API Composition

많은 MSA 요소가 그렇듯 역할 경계가 명확하지 않음

- Front Server, Aggregator, Mashup API, API Composition 등 다양하게 불림
- 차라리 CQRS로 검색하는 걸 추천

클라이언트 개발자님들이 고객

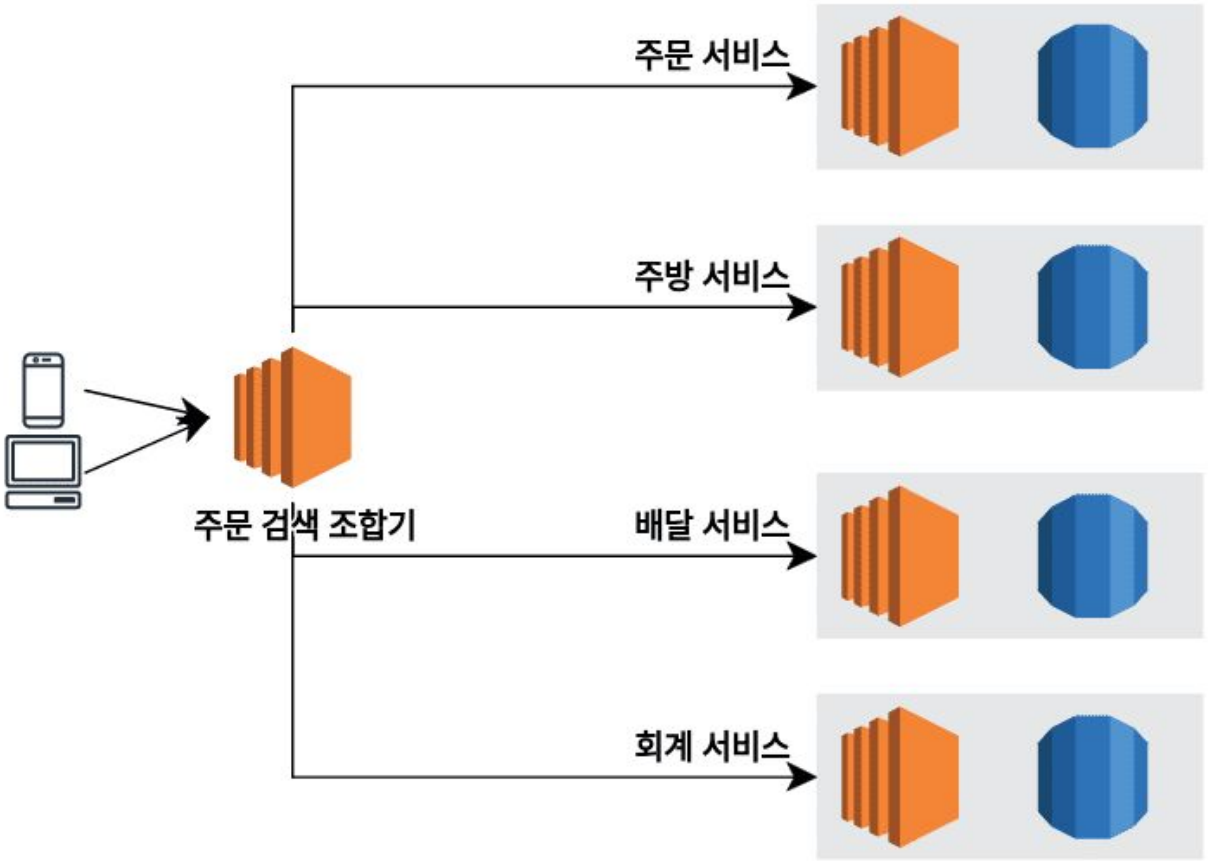
- 클라이언트가 직접 호출하는 거의 유일한 API
  - 클라이언트와 API 모두가 구현 가능한 비즈니스 로직? -> API 구현
- 모든 트래픽을 가장 먼저 맞음

API Composition

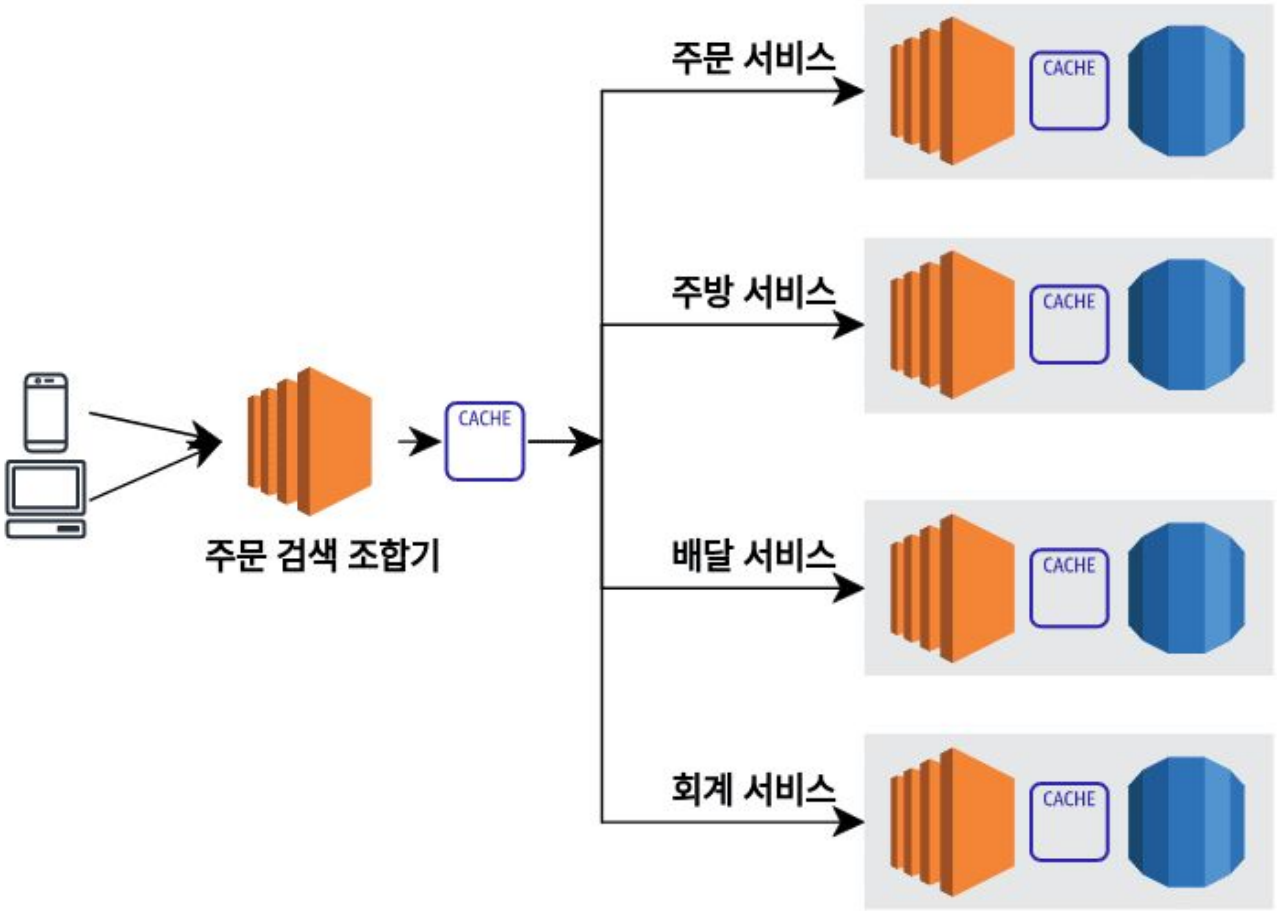
별도 서비스로 구현되는 경우가 많음

- API Gateway에서는 인증 정도만 처리하고 API Composition은 별도 서비스로 구현하는게 대부분

도입 후, 내부 서비스가 고통받는 구조



여기서 최대한 버텼으면...


주문 서비스
CACHE
주방 서비스
CACHE
CACHE
주문 검색 조합기
배달 서비스
CACHE
회계 서비스
CACHE

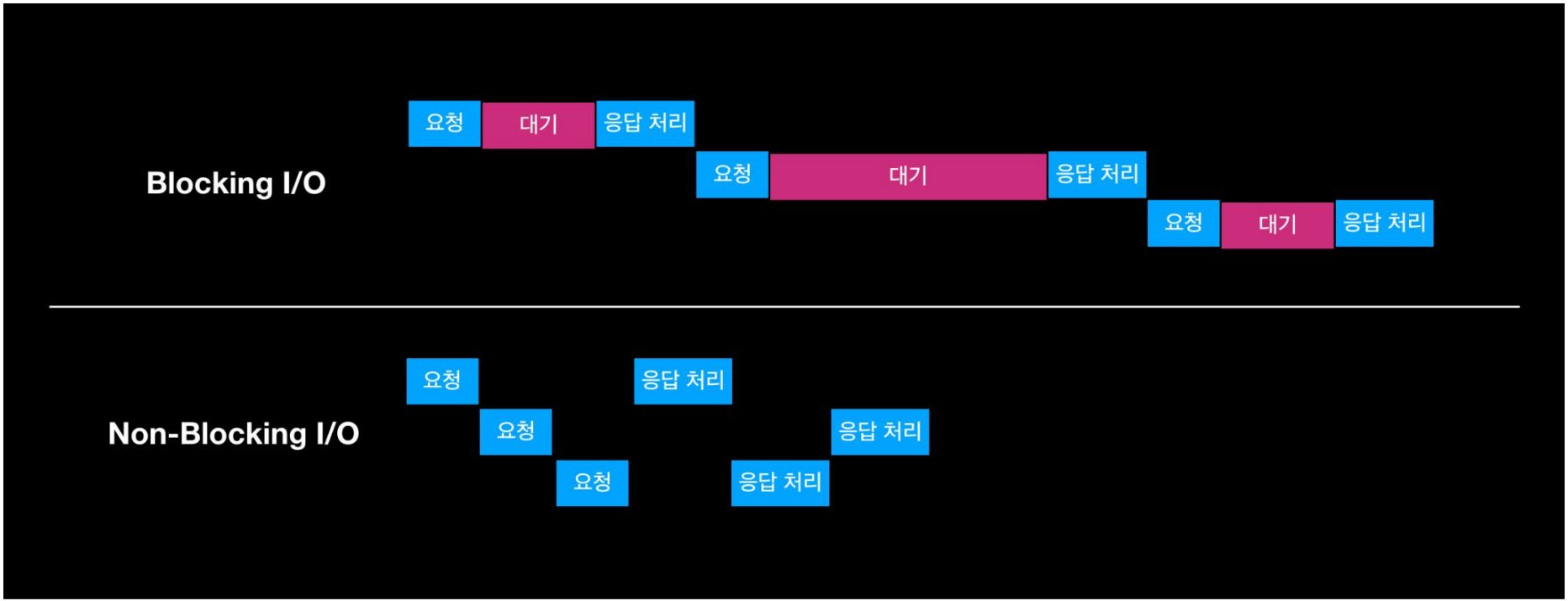
API Composition

Asynchronous, Non-Blocking I/O

- Spring WebFlux, Node.js로 많이 작업하는 이유

Asynchronous

Non-Blocking I/O



API Composition - 장점

시스템 디자인이 직관적

- 내부 서비스를 모두 호출 후 응답에 비즈니스 로직 더해서 제공

클라이언트 생산성 증대

- 호출 복잡도가 사라지고 집중해야 할 것(UI, 플랫폼 종속적 이슈 등)에 집중할 수 있도록 도움
- 네이티브 클라이언트의 업데이트 최소화를 도움

API Composition - 단점

구현할 수 없는 요구사항이 결국 발생

- 실시간 호출로 구현하기에는 너무 비효율적이거나 불가능한 요구사항이 발생

CQRS

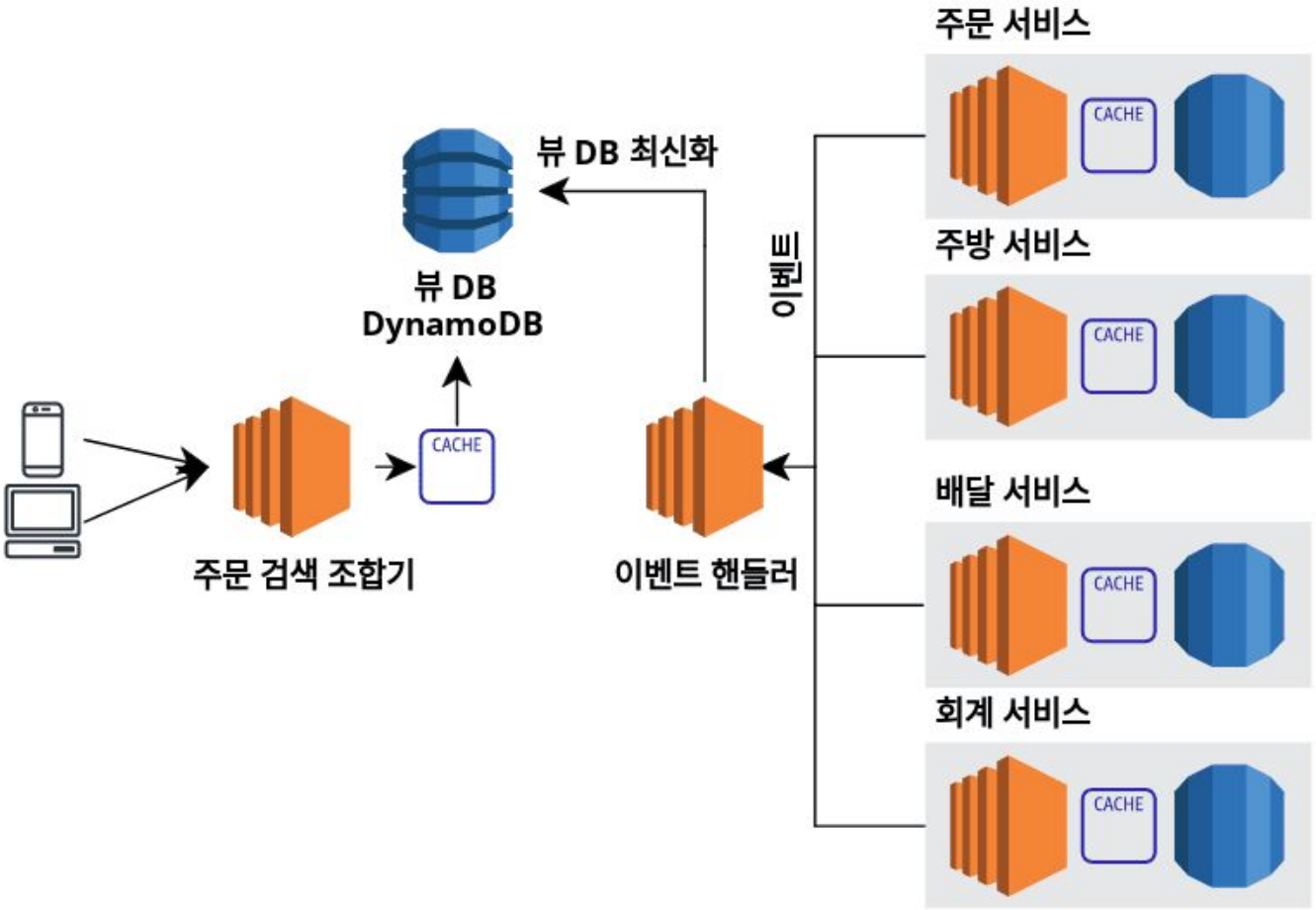
특징

다양한 뷰 설계 살펴보기

도메인과 레이어의 위치 등에 따라 천차만별

장단점

뷰 전용 DB 추가


주문 서비스
CACHE
주방 서비스
CACHE
배달 서비스
CACHE
회계 서비스
CACHE
이벤트
뷰 DB 최신화
뷰 DB
DynamoDB
CACHE
주문 검색 조합기
이벤트 핸들러

CQRS

CQRS는 용어 그대로 명령과 조회의 분리

- 명령과 조회를 분리한다는 컨셉만 지킬 뿐 실제 구현(특히 뷰 설계 쪽)은 매우 다양함
- 캐싱 전략이 Read-throught와 Write-throught 섞인 경우가 많음
- 조회의 트래픽이 명령을 수행하는 내부 서비스에 전파되지 않는 것이 핵심 아닐까?
    - API Composition과 달리 쿼리 데이터의 종단이 내부 서비스가 아닌 자체 뷰 DB

뷰 전용 DB

반정규화된 데이터 저장

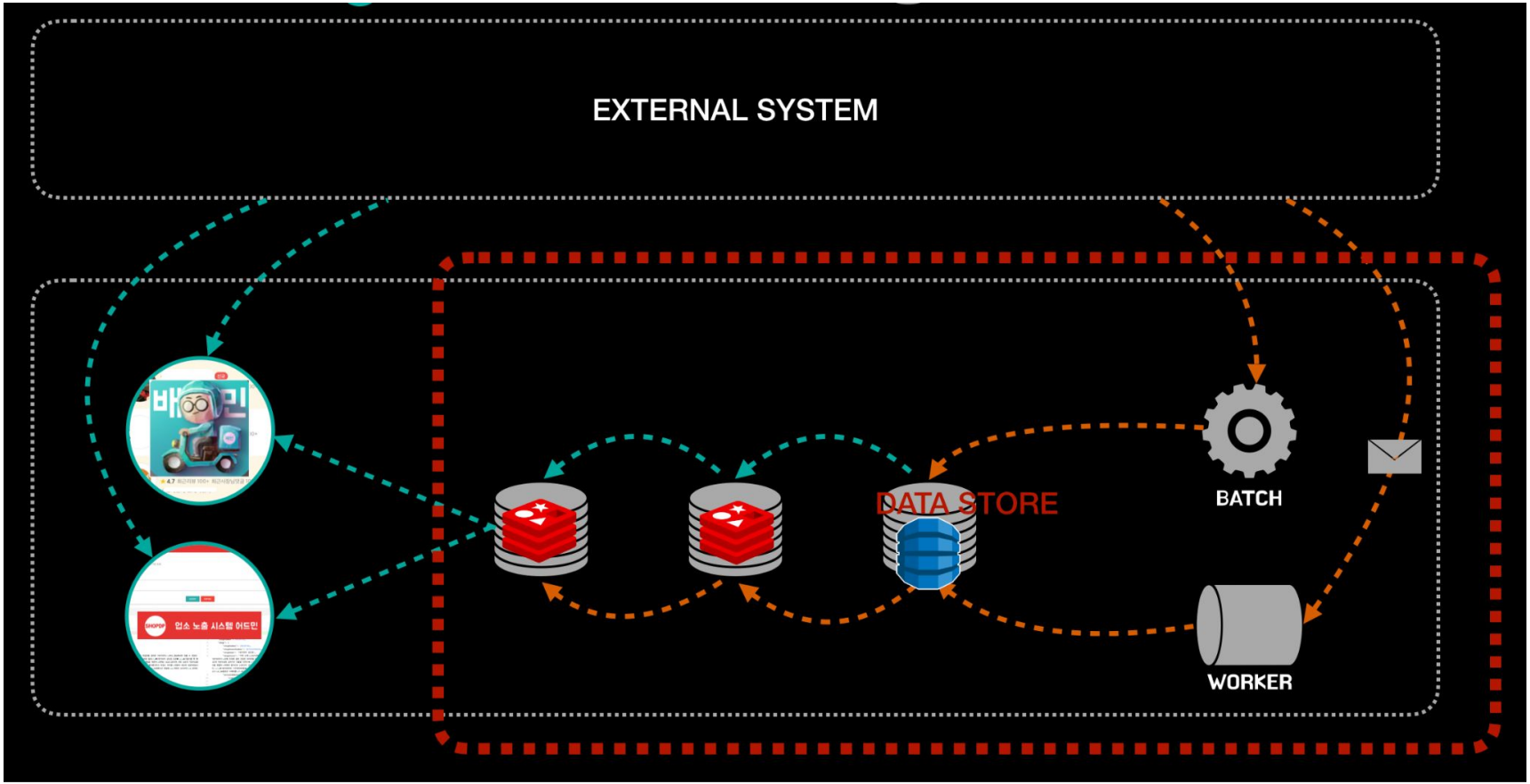
- (아이러니하게도) 잘 정규화 시켜놓은 데이터들을 역으로 반정규화해서 저장하는 DB
- DynamoDB, MongoDB 등이 많이 사용되지만 RDBMS도 나쁘지 않다는 생각
  - 조회 조건이 단순하고 응답 JSON을 그대로 저장하는 경우가 많기 때문
  - MySQL JSON vs. TEXT :
- 필터링 조건이 많은 목록 제공 필요 시 ElasticSearch도 뷰 DB로 사용하고 있음

최종적 일관성 추구

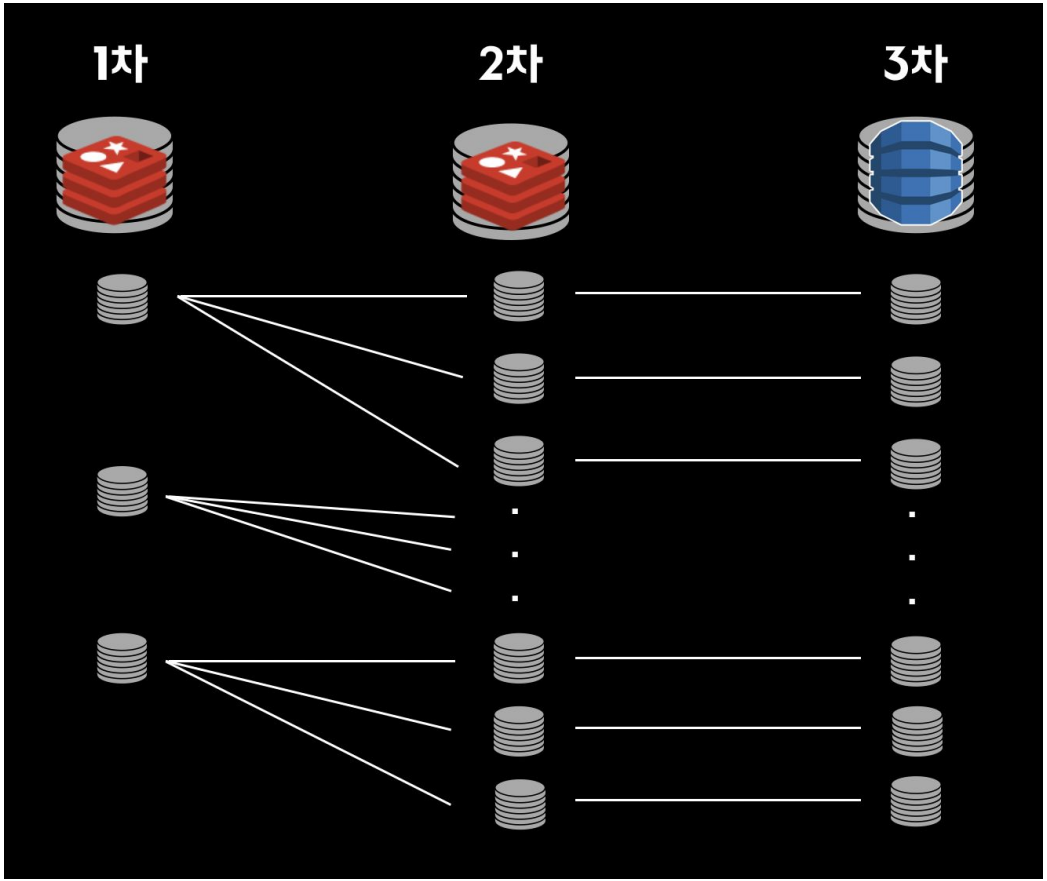
애초에 일관성에 대한 요구가 느슨한 쿼리만 제공해야...

- 시스템 디자인의 목적은 장애가 절대 발생하지 않는 시스템이 아닌 장애 발생 시 빠르게 복구 가능한 시스템
- 중복 이벤트 솎아 내기, 동시 업데이트 처리 등은 복잡한 아키텍쳐에서 복잡도를 더욱 높이는 일
- 클라이언트가 뷰의 최종 일관성을 처리하는건 더욱 드문 일, 가능하면 하지 않았으면...

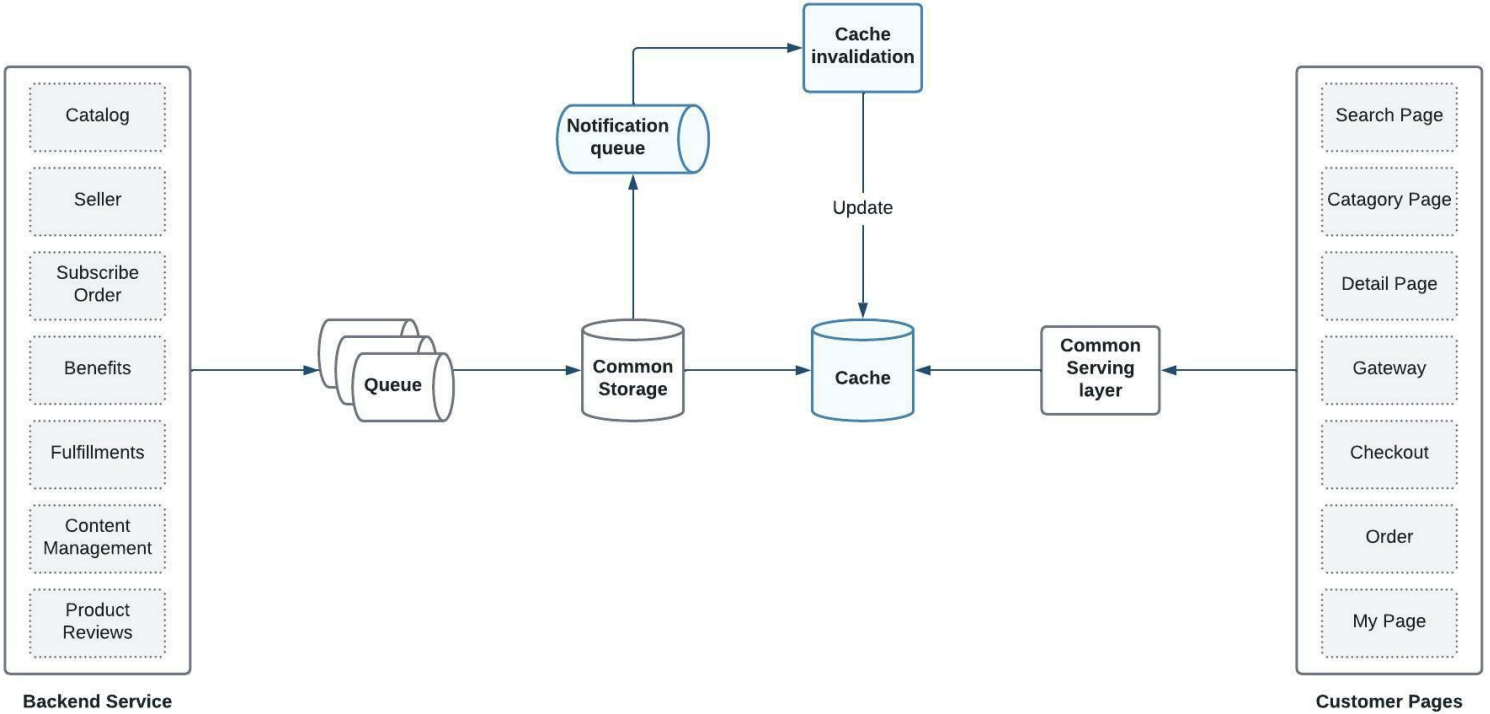
배달의 민족 가게 노출 시스템



배달의 민족 가게 노출 시스템



쿠팡


Catalog
Seller
Subscribe Order
Benefits
Fulfillments
Content Management
Product Reviews
Backend Service
Queue
Common Storage
Notification queue
Cache invalidation
Update
Cache
Common Serving layer
Search Page
Catagory Page
Detail Page
Gateway
Checkout
Order
My Page
Customer Pages

CQRS 장단점

API Composition으로 불가능한 쿼리 구현 가능

읽기(쿼리) 성능 최대화

- 쿠팡의 경우 다양하게 고도화 하여 120~200개의 Redis node로 1억회/분의 요청을 소화한다 함

아키텍쳐의 복잡성 상승

- 이벤트 기반으로 뷰 모델을 관리하는게 보통이고 캐싱 정책을 고도화 할수록 아키텍쳐가 복잡해짐
- 다양한 failover 정책 및 대응 시나리오 필요

CQRS 이후

BFF(Backend For Frontend)

- 쿼리 API 레벨에서의 고도화

캐싱, 정적으로 제공할 수 있는 컨텐츠 API와 분리

- API 호출을 줄이기 위한 끊임없는 최적화

참고자료

- Netflix API 아키텍처 진화
- 배달의 민족 사례
  - B마트 전시 도메인 CQRS 적용하기
  - [우아콘2020] 배민 프론트서버의 사실과 오해
  - 배달의민족 최전방 시스템! '가게노출 시스템'을 소개합니다.
- 쿠팡 사례
  - 대용량 트래픽 처리를 위한 쿠팡의 백엔드 전략
  - 캐시를 활용한 대용량 트래픽 처리 성능 향상
- 오늘의집 MSA Phase 1. API Gateway
- CQRS 키워드로 검색하시면 무궁무진하게 많습니다.

감사합니다