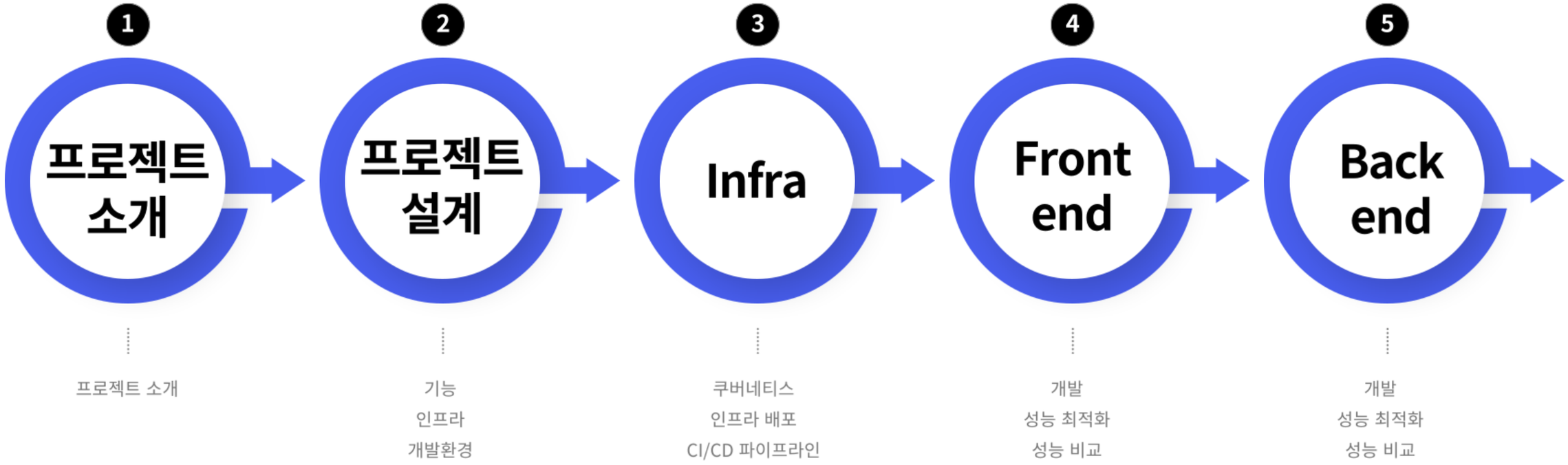
NO.1 택배 운송장 정보보호 서비스

EXQRESS

이제 걱정하지 마세요!
개인정보 걱정, EXQRESS가 해결해 드리겠습니다.

EXQRESS

목차


1
프로젝트 소개
프로젝트 소개
2
프로젝트 설계
기능
인프라
개발환경
3
Infra
쿠버네티스
인프라 배포
CI/CD 파이프라인
4
Front end
개발
성능 최적화
성능 비교
5
Back end
개발
성능 최적화
성능 비교

EXQRESS

제거

경기도 의정부시 오목로 150 211동 1203호

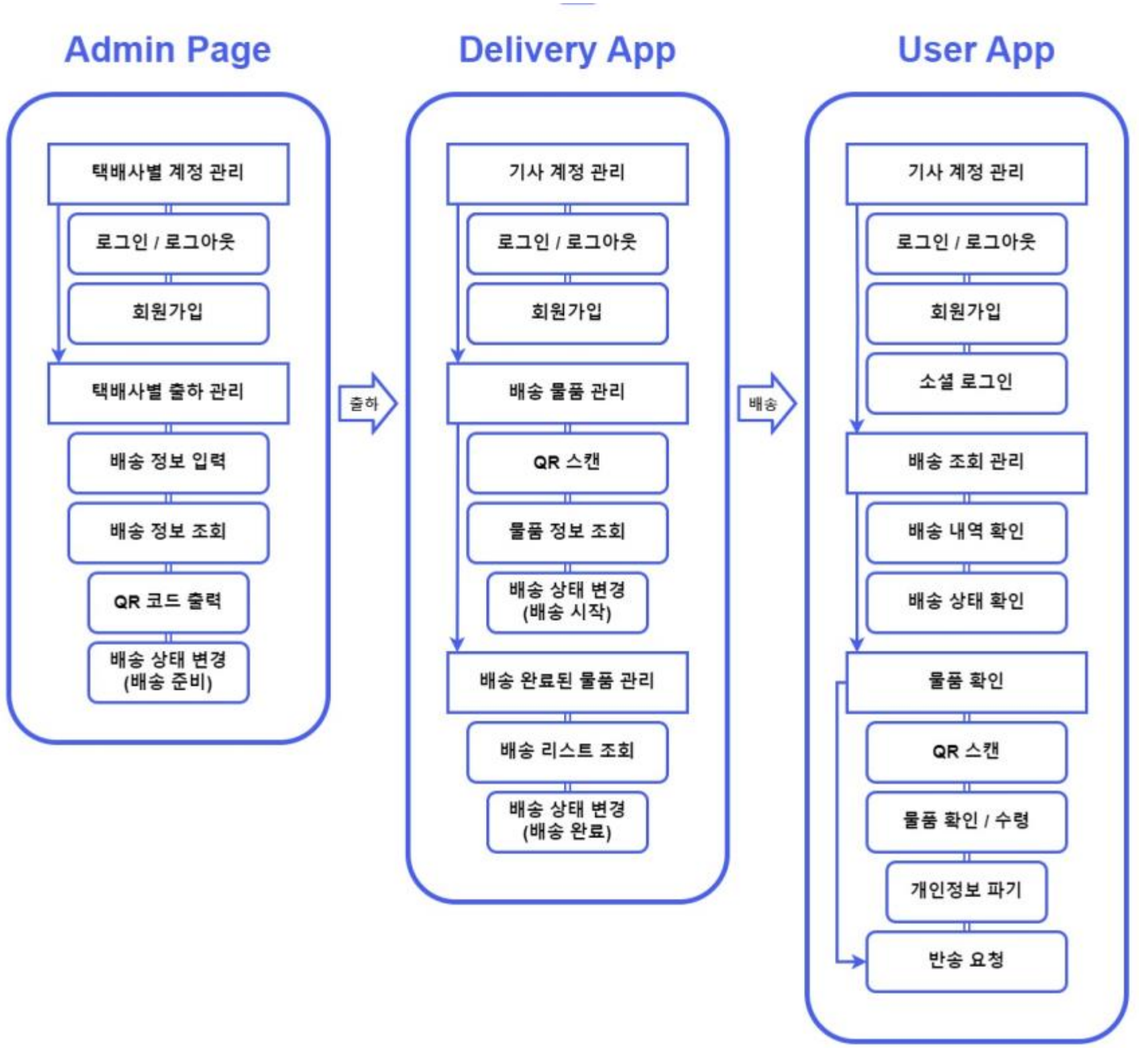
물품명 :

데일리 트레이닝 와이드팬츠(4 Color)

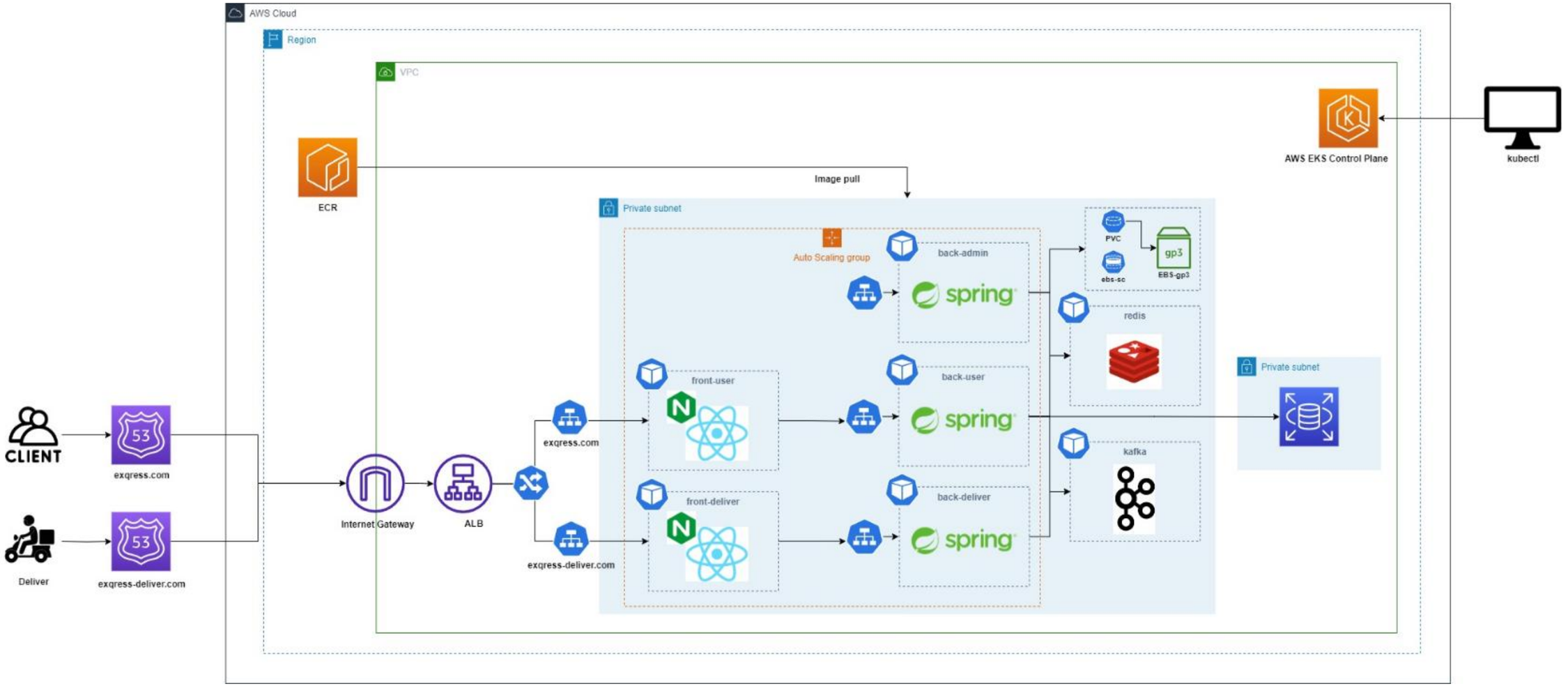
프로젝트 개발 절차/계획

추진 내용	1주차	2주차	3주차	4주차	5주차	6주차	7주차	8주차	9주차	10주차	11주차1	12주차	13주차	14주차	15주차
프로젝트 설계															
인프라 개발															
Front 개발															
Backend 개발															
프로젝트 기능/배포 테스트															
최종 발표															

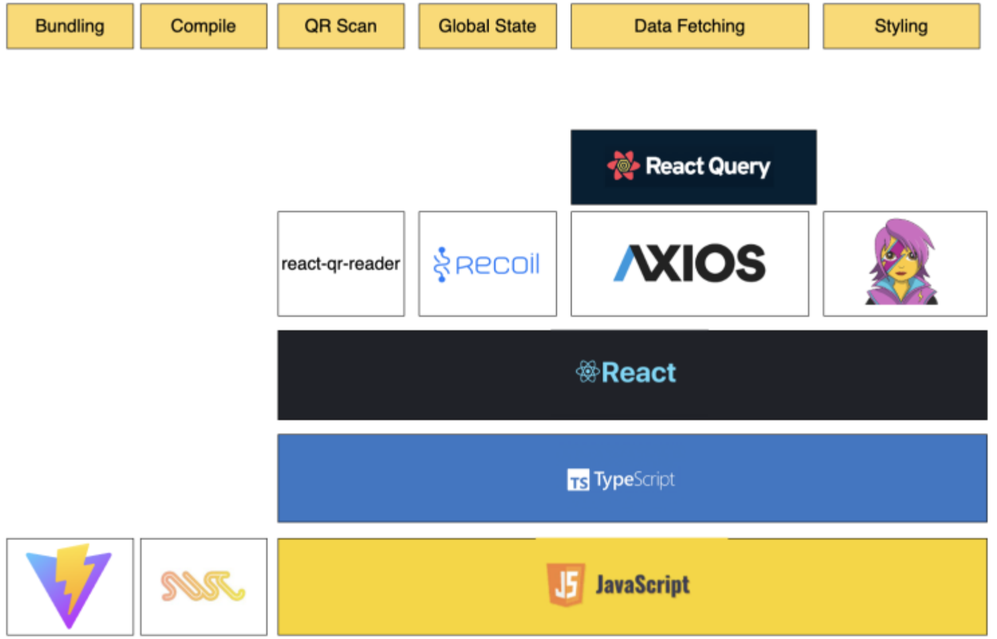
프로젝트 설계 - 기능

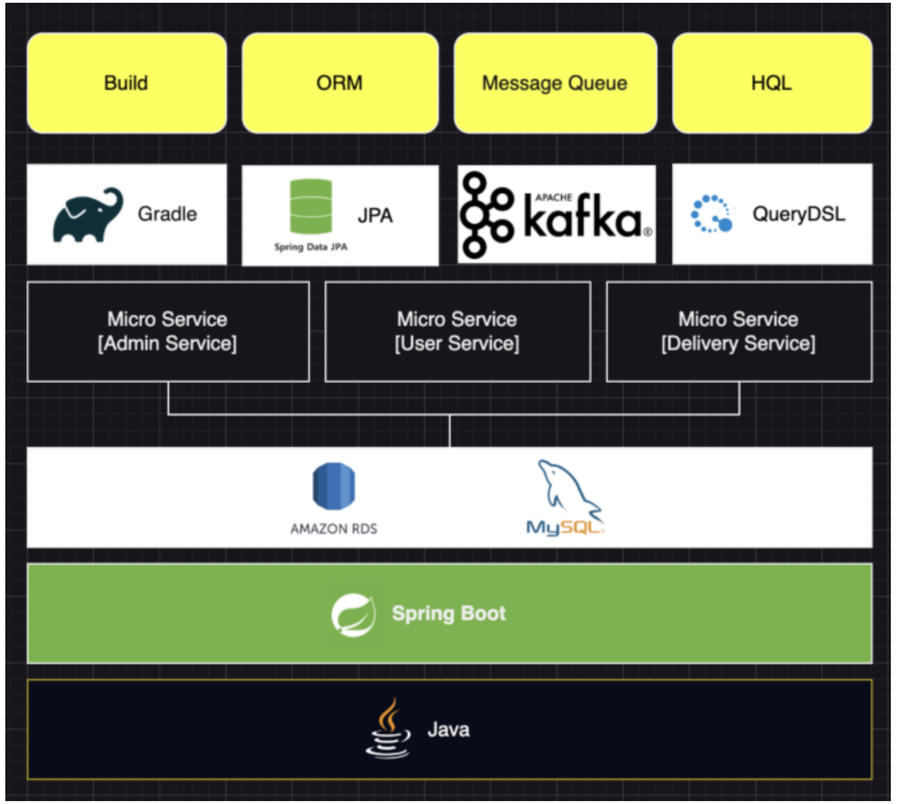

Admin Page
택배사별 계정 관리
로그인 / 로그아웃
회원가입
택배사별 출하 관리
배송 정보 입력
배송 정보 조회
QR 코드 출력
배송 상태 변경
(배송 준비)
출하
Delivery App
기사 계정 관리
로그인 / 로그아웃
회원가입
배송 물품 관리
QR 스캔
물품 정보 조회
배송 상태 변경
(배송 시작)
배송 완료된 물품 관리
배송 리스트 조회
배송 상태 변경
(배송 완료)
배송
User App
기사 계정 관리
로그인 / 로그아웃
회원가입
소셜 로그인
배송 조회 관리
배송 내역 확인
배송 상태 확인
물품 확인
QR 스캔
물품 확인 / 수령
개인정보 파기
반송 요청

인프라 설계

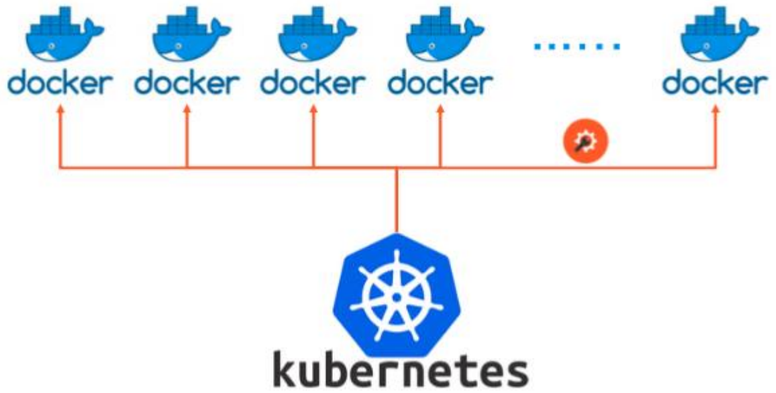

AWS Cloud
Region
VPC
ECR
Image pull
AWS EKS Control Plane
kubectl
Private subnet
Auto Scaling group
back-admin
spring
PVC
ebs-sc
gp3
EBS-gp3
redis
front-user
back-user
front-deliver
back-deliver
kafka
Private subnet
CLIENT
exqress.com
Deliver
exqress-deliver.com
Internet Gateway
ALB
exqress.com
exqress-deliver.com

프로젝트 설계 - Frontend/Backend



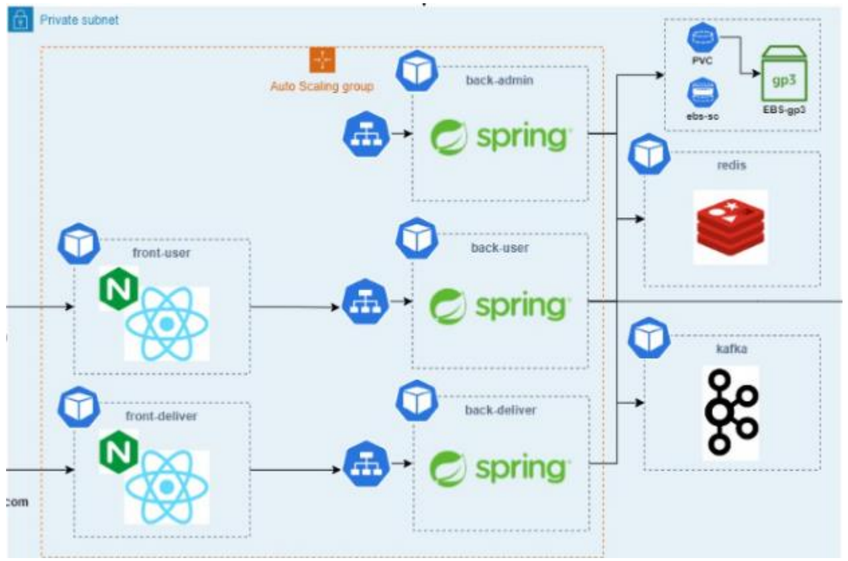


Kubernetes를 사용하여 배포



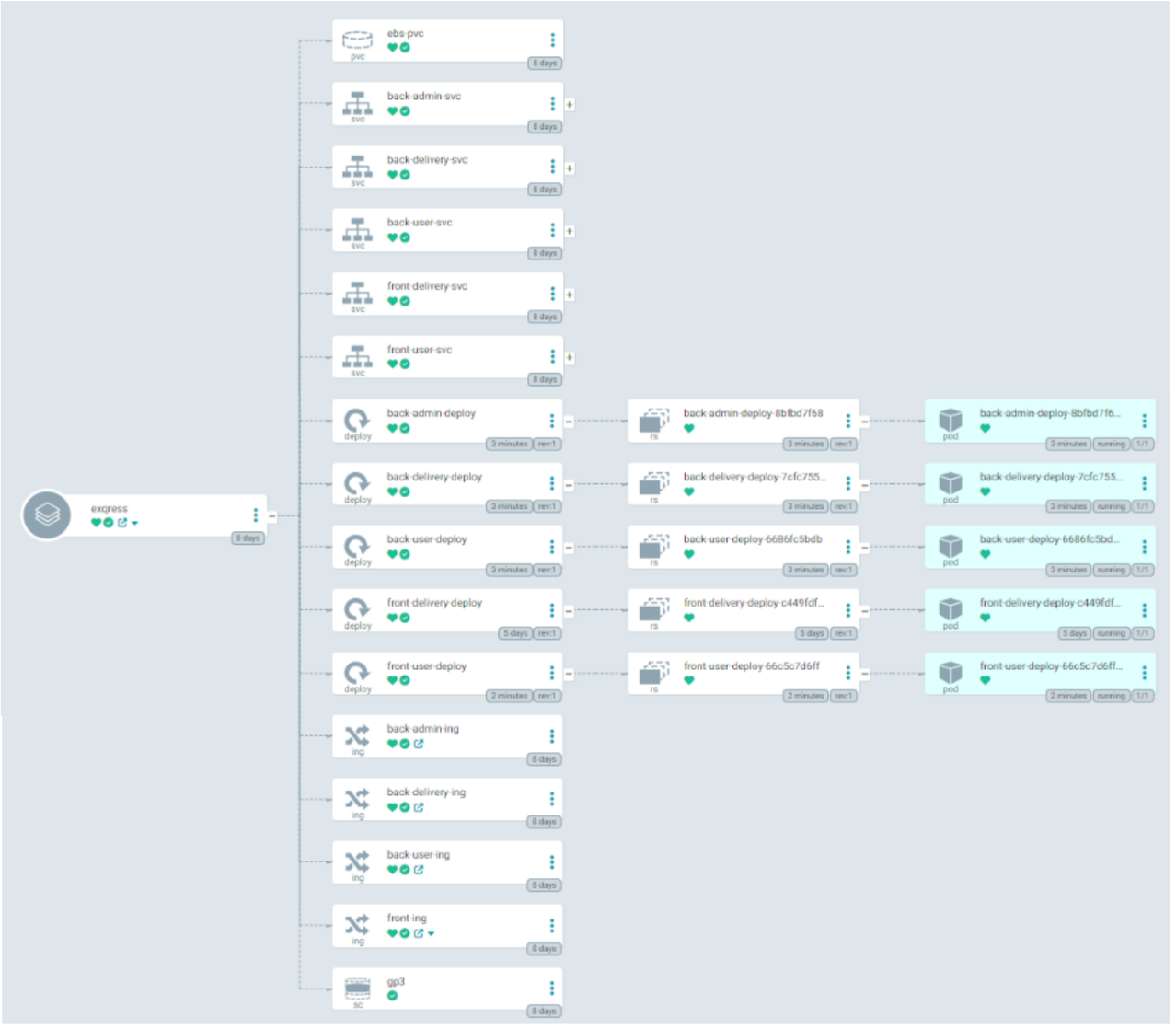
- 컨테이너화된 애플리케이션을 배포, 관리, 확장할 때 수반되는 다수의 수동 프로세스를 자동화하는 오픈소스 컨테이너 오케스트레이션 플랫폼
- 모든 서버를 개별적으로 배포/관리 하는 것은 어렵고 비효율적

- 쿠버네티스 서비스인 EKS에 모든 서버를 배포함으로써 관리 용이
- 내부 네트워크 통신망을 사용하여 서비스끼리 통신하기 때문에 트래픽 속도가 빠름



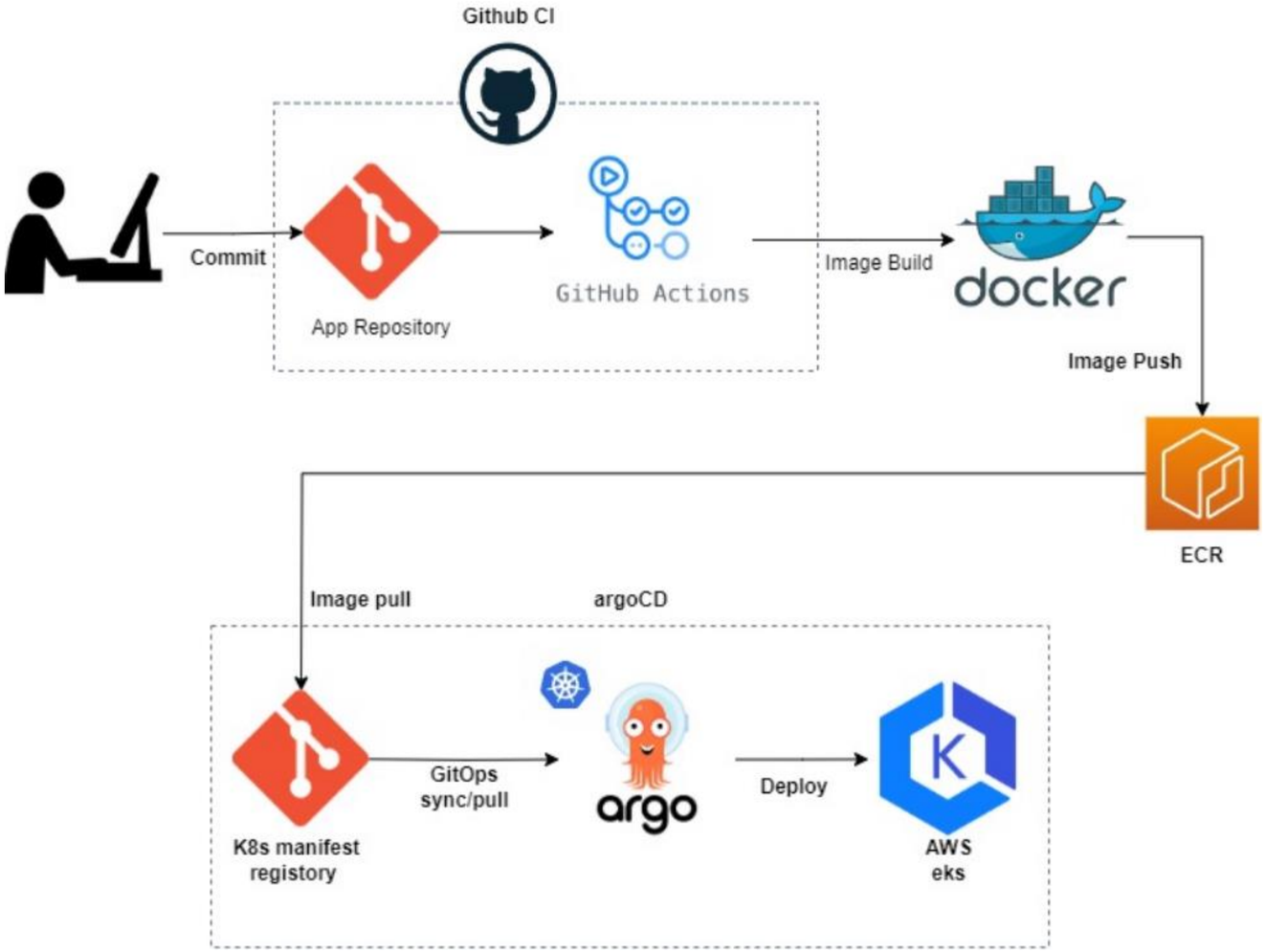
Infra 개발

	hsshin0602 Update image back-admin tag 17	`6d3191e` 2 hours ago
	.gitignore	add manifest
	README.md	first commit
	back-admin-deploy.yaml	Update image back-admin tag 17
	back-admin-ing.yml	add manifest
	back-admin-svc.yaml	modify admin
	back-delivery-deploy.yaml	Update image tag 57
	back-delivery-ing.yaml	add https
	back-delivery-svc.yaml	add manifest
	back-user-deploy.yaml	Update image tag 54
	back-user-ing.yaml	add manifest
	back-user-svc.yaml	add manifest
	ebs-pvc.yaml	add storage
	front-delivery-deploy.yaml	Update image tag 71
	front-delivery-svc.yaml	add manifest
	front-ing.yaml	a
	front-user-deploy.yaml	Update image tag 41
	front-user-svc.yaml	add manifest
	gp3-sc.yaml	a



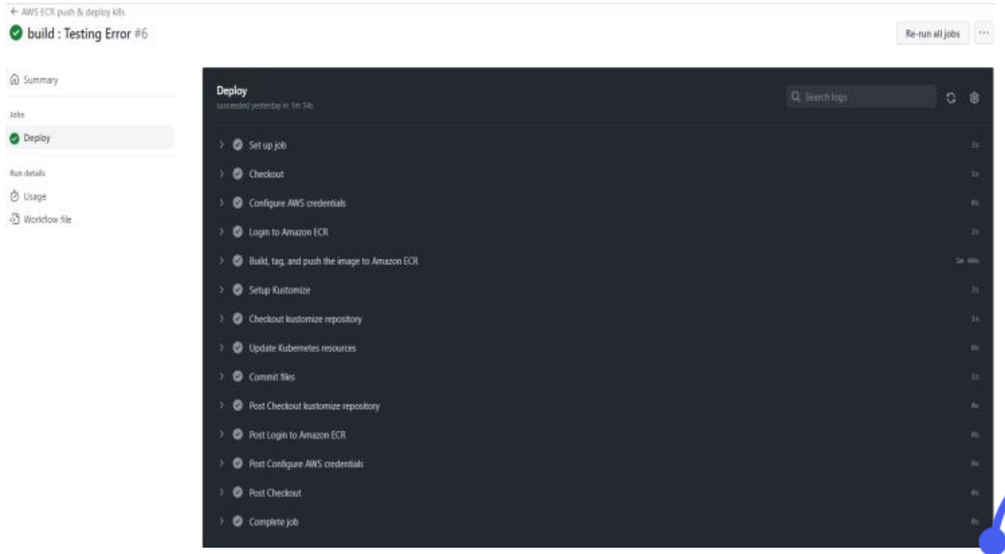
- k8s manifest 파일를 직접 작성하여 프로젝트 배포
- argoCD를 통해 배포 상태 관리

CI/CD 파이프라인 구축

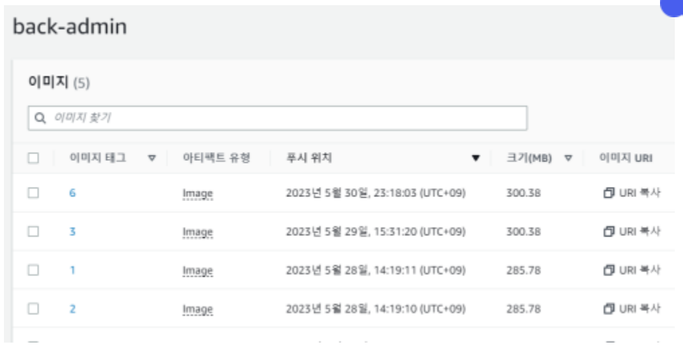


- 개발자가 git 저장소에 개발한 코드를 Push
- 생성된 이미지를 private ECR에 Push
- k8s용 git 저장소에서 이미지 change
- ArgoCD가 감지하여 새로운 버전으로 deploy

CI/CD 파이프라인 FLow

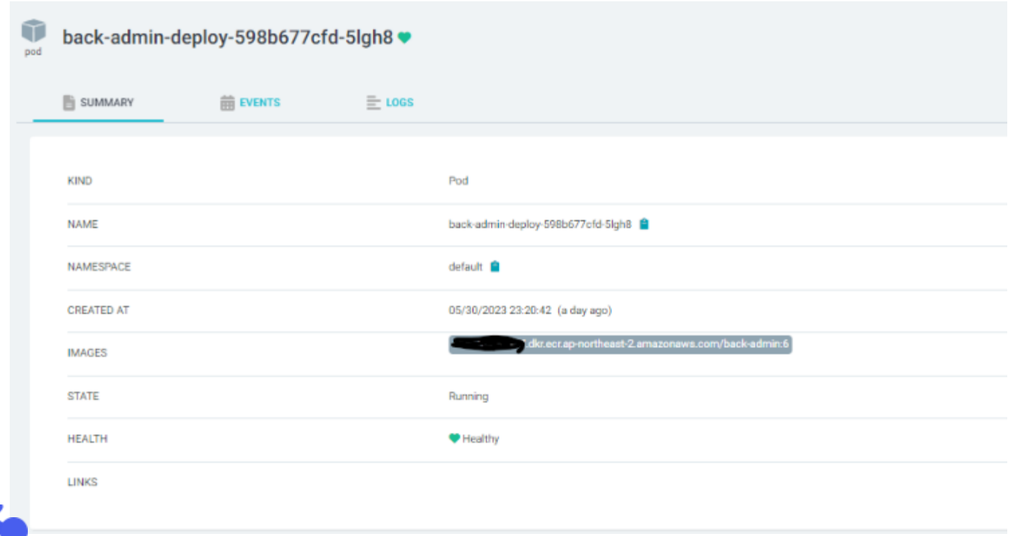


- Github action 빌드

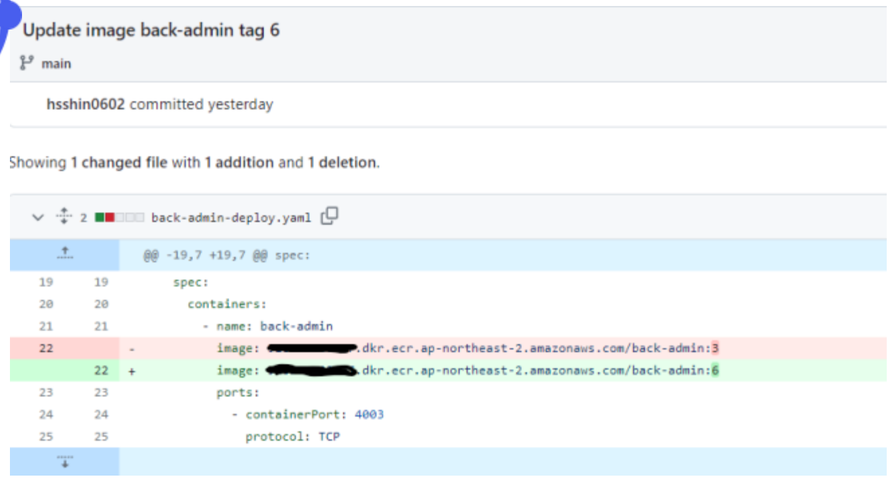


- ECR에 이미지 생성

- ArgoCD 배포



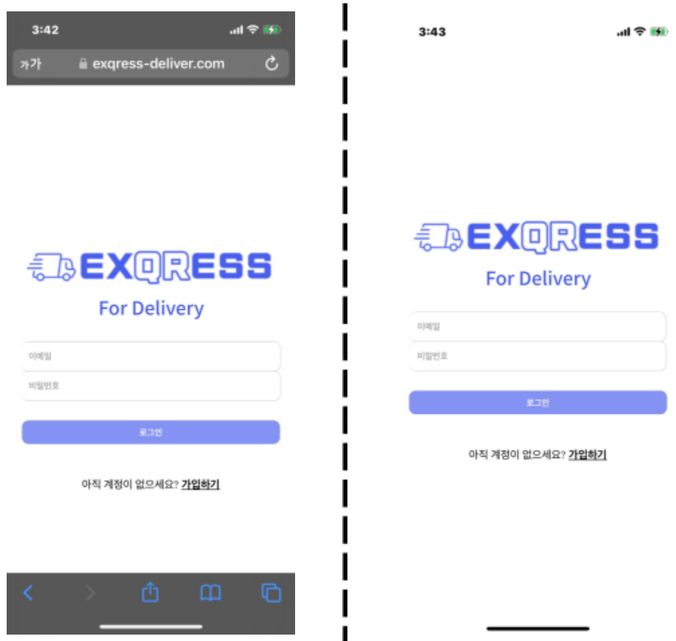
- k8s manifest 파일 저장소 이미지 교체



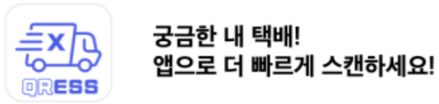
Frontend 개발

PWA를 사용한 앱 개발

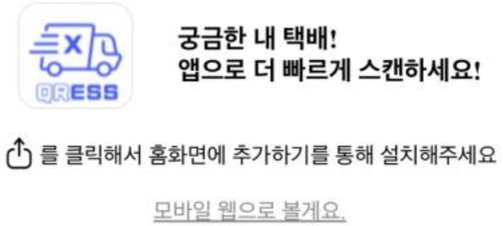
- 사용자와 기사가 QR 코드를 찍기 위해선 모바일 앱이 필요
- 짧은 기간동안 앱을 개발후 스토어에 배포하여 실제 작동을 확인하는 것은 불가능하다고 판단
- PWA에서도 주요 기능인 QR 스캔이 가능한 것을 확인



Web | App



Android



IOS

PWA 설치 안내

- 사용자가 PWA의 설치방법을 모를 경우에 대비해 설치 안내를 띄워 사용자들이 우리의 앱을 쉽게 설치할 수 있도록 설정

  Android: 설치버튼 / IOS: 설치안내

Frontend 성능 최적화

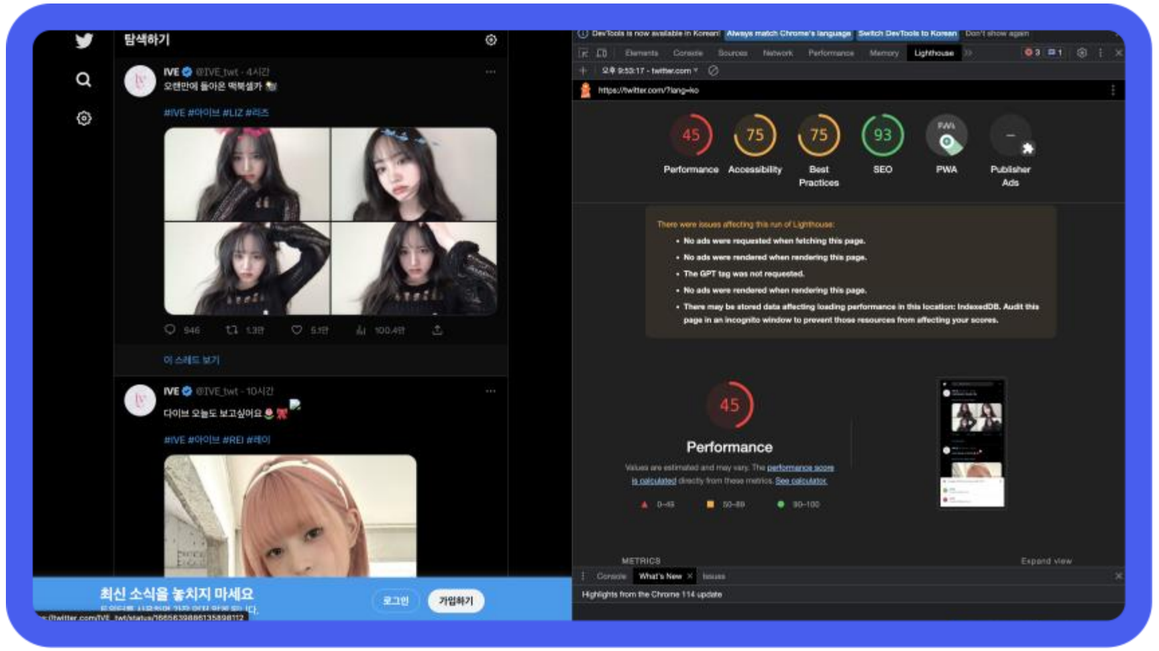
- https 적용후 HTTP/1.1에서 HTTP/2로 변경
- Nginx에 server push 기능을 추가하여 초기 브라우저 요청에 대한 응답 속도 항샹

- 택배정보를 불러오는 페이지에서 실시간으로 데이터를 받야와야 하기 때문에 불필요한 렌더링이 자주 발생
- 이를 React Query의 캐싱을 통해 데이터가 바뀐 경우에만 렌더링을 시키도록 함으로써 성능을 높일 수 있었다.

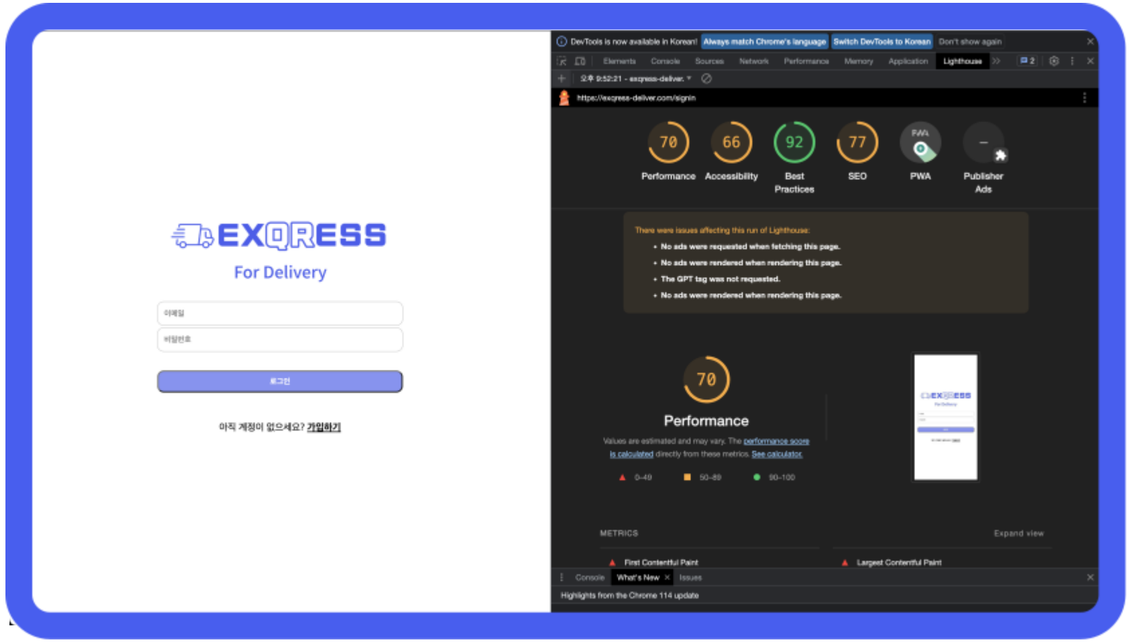
Frontend 성능 분석 비교

- 구글의 lighthouse(웹 앱의 성능, 품질 및 정확성을 개선하기 위한 오픈 소스 자동화 도구)를 통해 검사해본 결과
  PWA의 모든 조건을 만족하였고
  다른 PWA의 대표적 예시인 트위터와 비교해 괜찮은 성능을 나타내는 것으로 확인

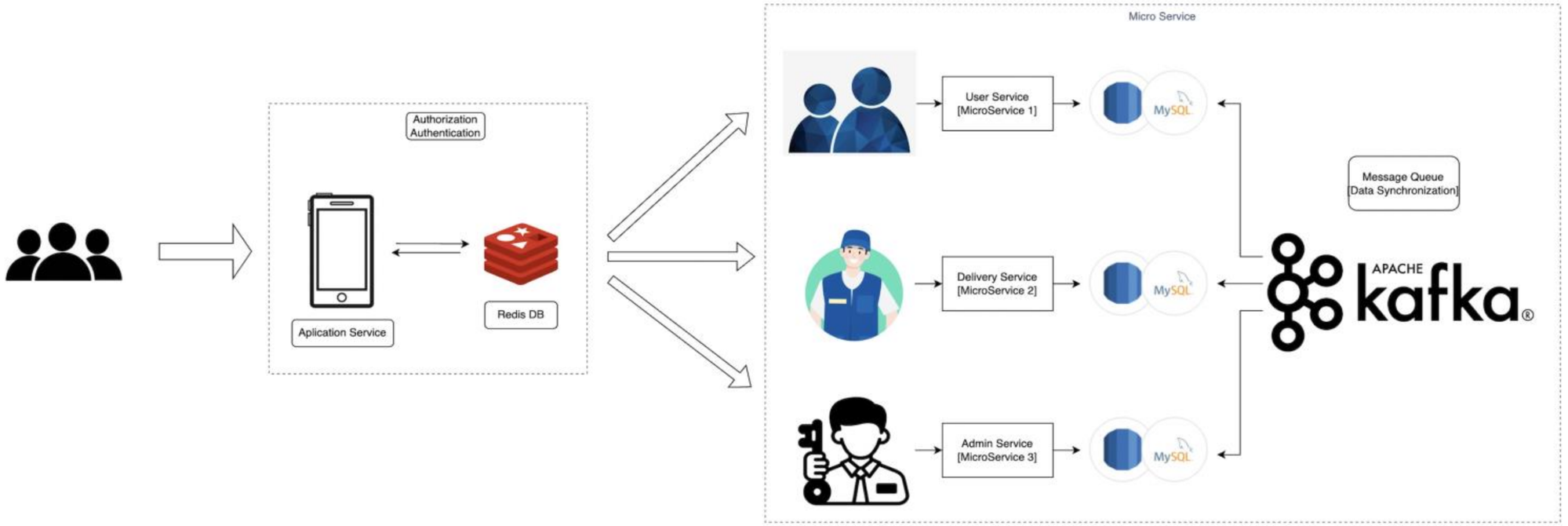
트위터



EXQRESS

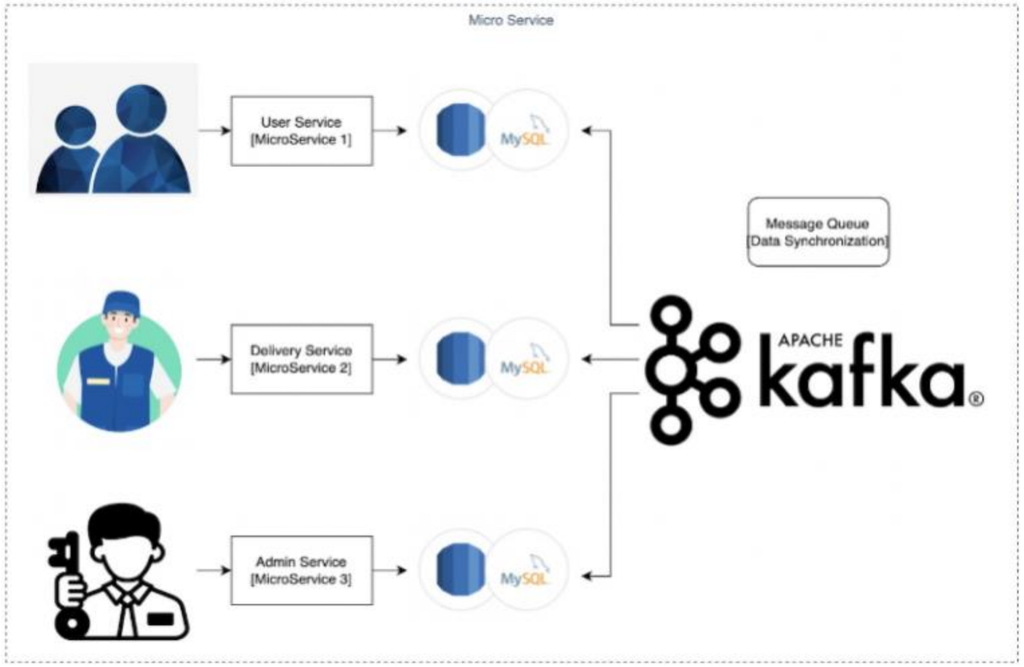


Backend 개발



- 사용자의 인증 처리(로그인) 기능은 휘발성 메모리 DB [Redis]를 사용하여 구현
- 기능 및 서비스 모듈화를 위해 자체적인 기능을 가지는 3개의 서비스로 나누어 구현
- 3개의 서비스는 모두 독립적인 DB로 유지 및 관리
- 모두 독립적인 DB지만 DB간 통신과 동기화 기능 요구 Apache Kafka 사용하여 데이터 동기화 처리

Kafka를 사용한 3개의 데이터베이스 동기화



Kafka란?

- 분산 스트리밍 플랫폼으로 데이터 스트림을 처리하고 저장하기 위한 기술
- 서비스 간 기능이 동작할 때마다 데이터 동기화 발생하여 서비스 간 확장성을 가능하게 하는 프로젝트 핵심적인 기술

Kafka 사용이유

- 다른 기능을 구현한 서비스 3개는 모두 독립적인 DB를 유지하지만 [택배 정보]라는 공통된 데이터는 3개의 서비스 간 동기화가 필수
- 택배에 붙어있는 QR 코드 안에는 Unique한 ID값만 존재하며 해당 값만을 이용하여 DB를 조회하여 정보를 요청
- 3개의 서비스는 [택배 정보]와 [택배 정보]를 조회하기 위해 연관된 [QR ID]값 동기화가 필수적

Backend 성능 최적화

Redis

- 휘발성 메모리 DB인 Redis를 사용하여 JWT 관리
- 휘발성 특징을 이용하여 정해진 시간이 지나면 자동으로 데이터(JWT) 정보가 삭제되도록 하여 DB 부하 줄이는 방법
- 뿐만 아니라 사용하지 않는 JWT는 자동 삭제가 되므로 보안 측면에서 장점도 가지고 있음

QueryDSL

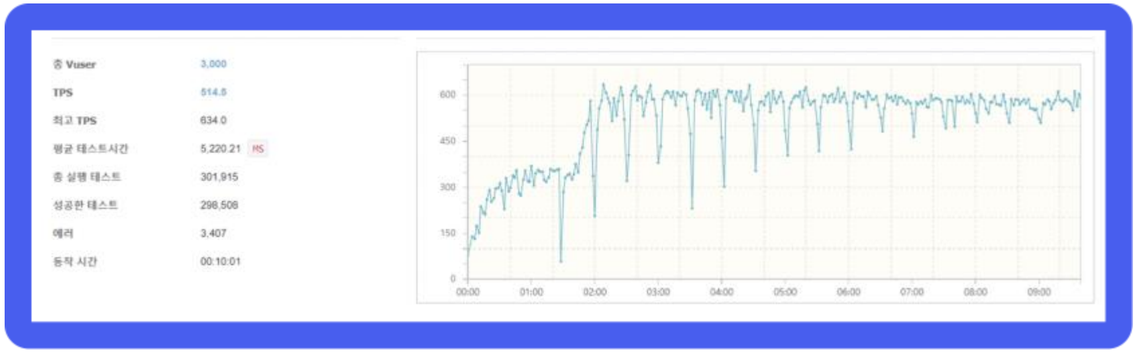
- JPA와 함께 사용하여 동적 쿼리를 생성
- 정적 쿼리가 아닌 동적 쿼리문을 작성하여 실행시킴으로써 DB 성능 최적화 동작을 가능하게 함
- Java build 과정에서 작성된 쿼리문에 대해 컴파일 오류를 발생시킴으로써 잘못된 쿼리문 작성을 하였을 경우 바로 잡을 수 있음

Backend 성능 분석 비교

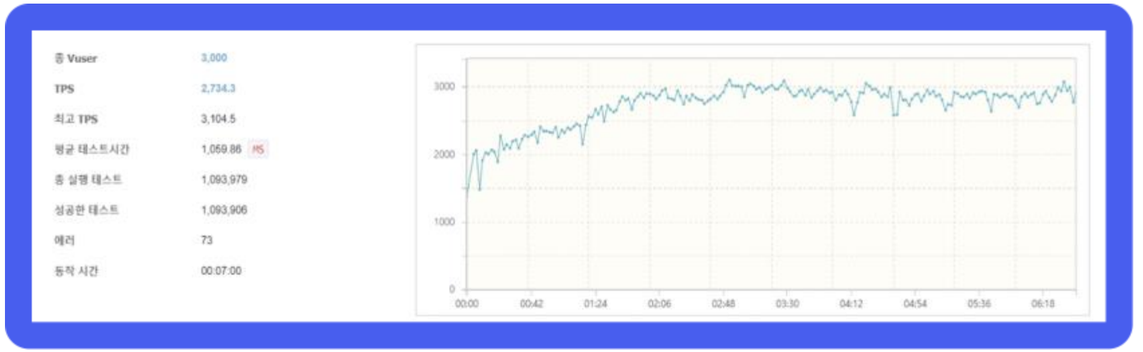
[POST - 로그인]

- Naver에서 개발한 백엔드 성능 테스트 오픈소스 Tool인 Ngrinder를 사용하여 백엔드 기능 성능 테스트 진행
- Ngrinder는 OS에 구애받지 않는 독립적인 환경에서 테스트를 진행하기 때문에 OS 튜닝은 이루어지지 않는 것으로 가정하여 진행하고 API와 DB 테이블 관계가 크게 복잡하지 않기 때문에 목표 TPS는 초기 서비스 목표값과 동일한 1000으로 설정
  [Compact] 1vCPU, 2GB Memory, 50GB 급 WAS Server 1대, MySQL DB환경]

POST Method



- 단일화 배포



- 이중화 배포

- 고도화 전 목표 TPS에 달성하지 못하는 평균 600대 값 기록
- 이중화 배포를 진행한 후 약 2배 넘는 1200대 값으로 최적화

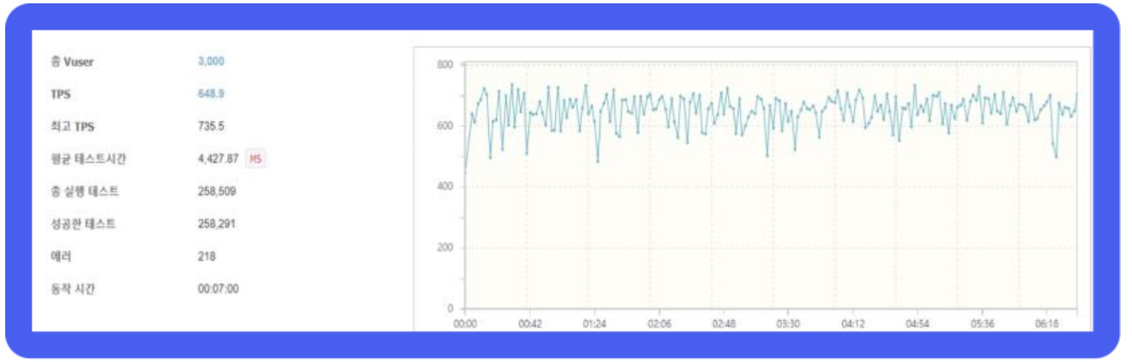
Backend 성능 분석 비교

[GET - 배송조회]

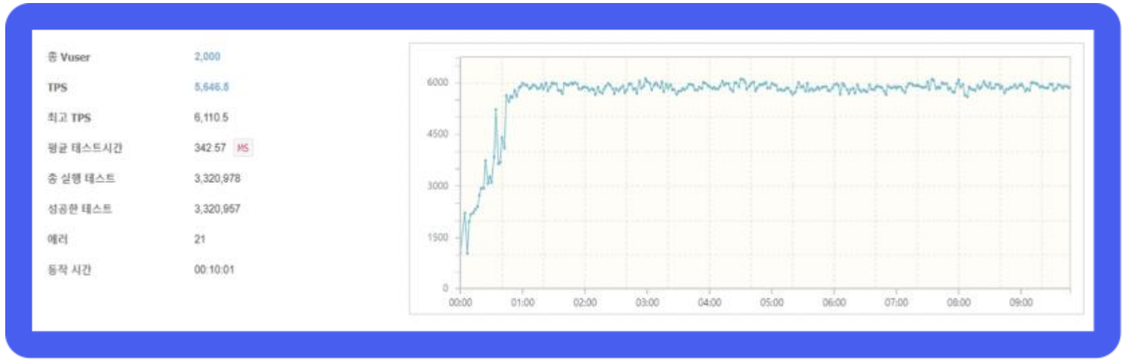
- 고도화 목표 전 TPS에 달성하지 못하는 650대 값 기록
- 이중화 배포와 QueryDSL 동적 쿼리문을 작성하여 진행한 후 약 2000대 값으로 최적화 -> 단순 조회인 Get방식이면서 QueryDSL로 동적 쿼리문까지 적용하여 큰 최적화를 볼 수 있었음

GET Method

- 단일화 배포



- 이중화 배포



EXQRESS

감사합니다.