



# CoinAlarm

하이브리드 클라우드를 이용한 메세지 발송 이중화

박용우, 지윤석



# 팀원 소개

## 팀장

박용우

## Part

Back-end, 프라이빗/퍼블릭 인프라 구축,  
CI/CD

Spring Boot

Kafka

CI/CD

K8s

AWS

## 팀원

지윤석

## Part

Front-end, Monitoring, 가격감지 서버

JavaScript

REACT

ExpressJS

K8s

AWS

# 목차

01. 서비스 ↗

02. 개발 ↗

03. 인프라 ↗

04. 모니터링&테스트 ↗

05. 회고 ↗

06. Q&A ↗

## 01. 서비스

### - 서비스 소개

실시간 코인 알림 서비스

#### 기존 거래소 알림 서비스

- 특정 시간대마다 알림을 받는 거래소 존재하지 않음
- 푸시알림으로 제공하기에 앱설치 필수



#### CoinAlarm 서비스

- 특정시간대 알림, 실시간 시세 등 다양한 기능 별 알림
- 전화번호 인증로 간편하게 SMS 알림

## 01. 서비스

### - 서비스 기능

#### 실시간 시세 알림



사용자의 원하는 목표가  
달성

#### 변동률 알림



사용자가 지정한 변동률을 달  
성시 알림

#### 특정 시간대 알림



특정한 날, 특정 시간마다 실  
시간 시세 알림

# 목차

01. 서비스 ↗

02. 개발 ↗

03. 인프라 ↗

04. 모니터링&테스트 ↗

05. 회고 ↗

06. Q&A ↗

## 02. 개발

### Projects Keyword

Prometheus  
Spring Boot  
Harbor Kafka Gitlab  
ArgoCD K8S Grafana  
MariaDB Docker Redis  
Jenkins ExpressJS  
Terraform

## 02. 개발

### - 협업

The screenshot illustrates the integration of JIRA and Notion for development collaboration. On the left, the JIRA backlog shows various stories (COIN-4 through COIN-64) with their descriptions and due dates. In the center, an API documentation page for 'API 명세' (API Specification) is displayed, featuring a '설명' (Description) section with '[notion]' highlighted. On the right, a detailed view of issue COIN-52, titled '[COIN-52] CloudFront 오류 인증서 관리' (CloudFront error certificate management), is shown. This view includes sections for '이유' (Reason), '주요 원인' (Major causes), and '해결 방법' (Solution methods), along with a list of related issues.

JIRA 를 통해 스토리카드에 에픽과 일정 관리  
노션링크로 들어가면 해당 이슈에 대한 자세한 내용을 정리

## 02. 개발

### - 협업



#### Git Commit Convention

##### Git Commit Message Convention

###### 커밋 메시지 컨벤션

1. JIRA 타겟 보드 번호 작성
2. 커밋 유형 지정
3. 제목과 본문을 빈행으로 분리
4. 제목 첫 글자는 대문자로, 끝에는 . 급지
5. 제목은 영문 기준 50자 이내로 할 것
6. 자신의 코드가 직관적으로 바로 파악할 수 있다고 생각하지 말자
7. 여러가지 항목이 있다면 글머리 기호를 통해 가독성 높이기
- 규칙에 맞는 좋은 커밋메시지를 작성해야 하는 이유
- 한 커밋에는 한 가지 문제만!
- CLI에서 커밋 메시지 여러 줄로 작성하는 방법

##### Git Commit Message Convention

###### 커밋 메시지 컨벤션

1. JIRA 타겟 보드 번호 작성

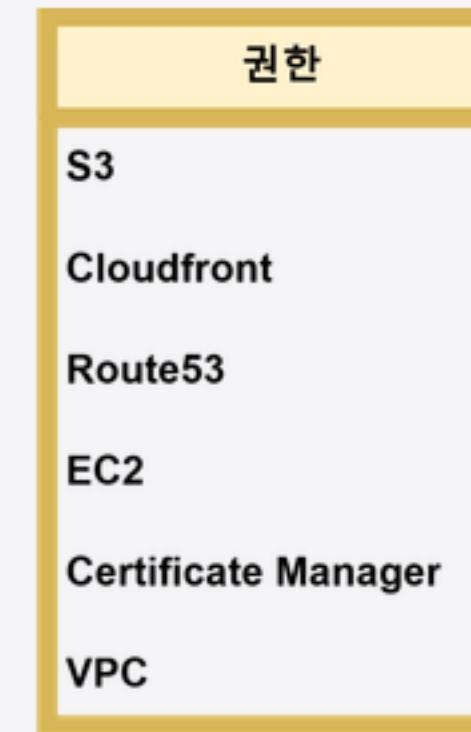
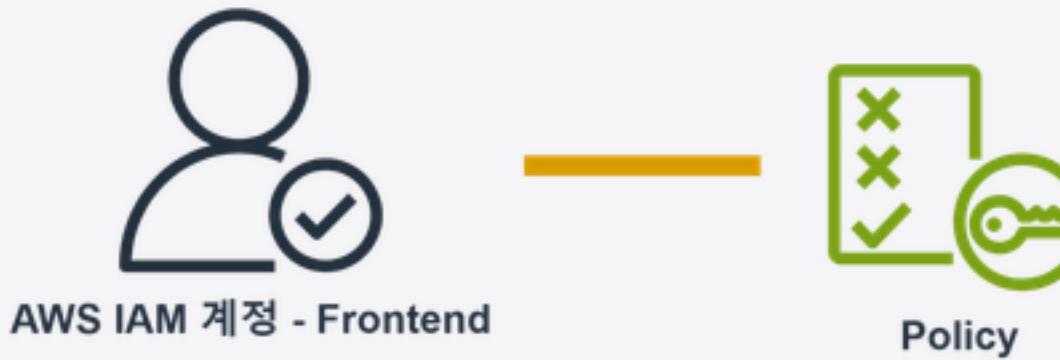
이름	최근 커밋
..	
CoinModal.jsx	[COIN-66] Design : 모달창 사이즈 수정
CoinTable.jsx	[COIN-36] Feat :로그인 로직 추가
ConfirmationModal.jsx	[COIN-66] Design : 모달창 사이즈 수정
Footer.jsx	[COIN-76] Feat : 푸터 추가
Graph.jsx	[COIN-56] Design : 그래프 css 조절 및 크기조정
Header.jsx	[COIN-24] Design : 헤더 스크롤 이벤트 추가
PatchModal.jsx	[COIN-66] Design : 모달창 사이즈 수정
RisingFallingCoins.jsx	[COIN-75] Fix : 웹소켓 로직 변경
ServiceTitle.jsx	[COIN-71] Refactor : 서비스타이틀 컴포넌트 분리
SubReservation.jsx	[COIN-75] Fix : 웹소켓 로직 변경
landingCoinList.jsx	[COIN-71] Refactor : 서비스타이틀 컴포넌트 분리
useScrollIndicator.jsx	[COIN-79] Fix : 비디오 애니메이션 에러 처리

코드 컨벤션을 정하여 서로 커밋메세지를 통해 편리하게 소통 가능

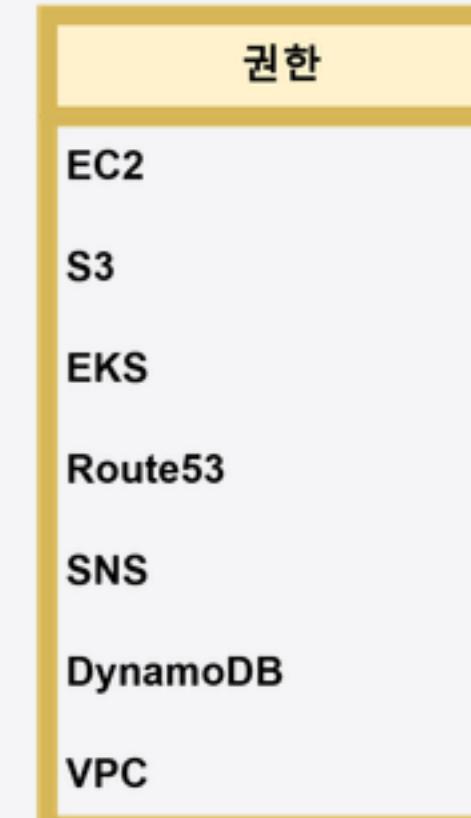
편리하게 과거 추적 가능

## 02. 개발

### - 협업



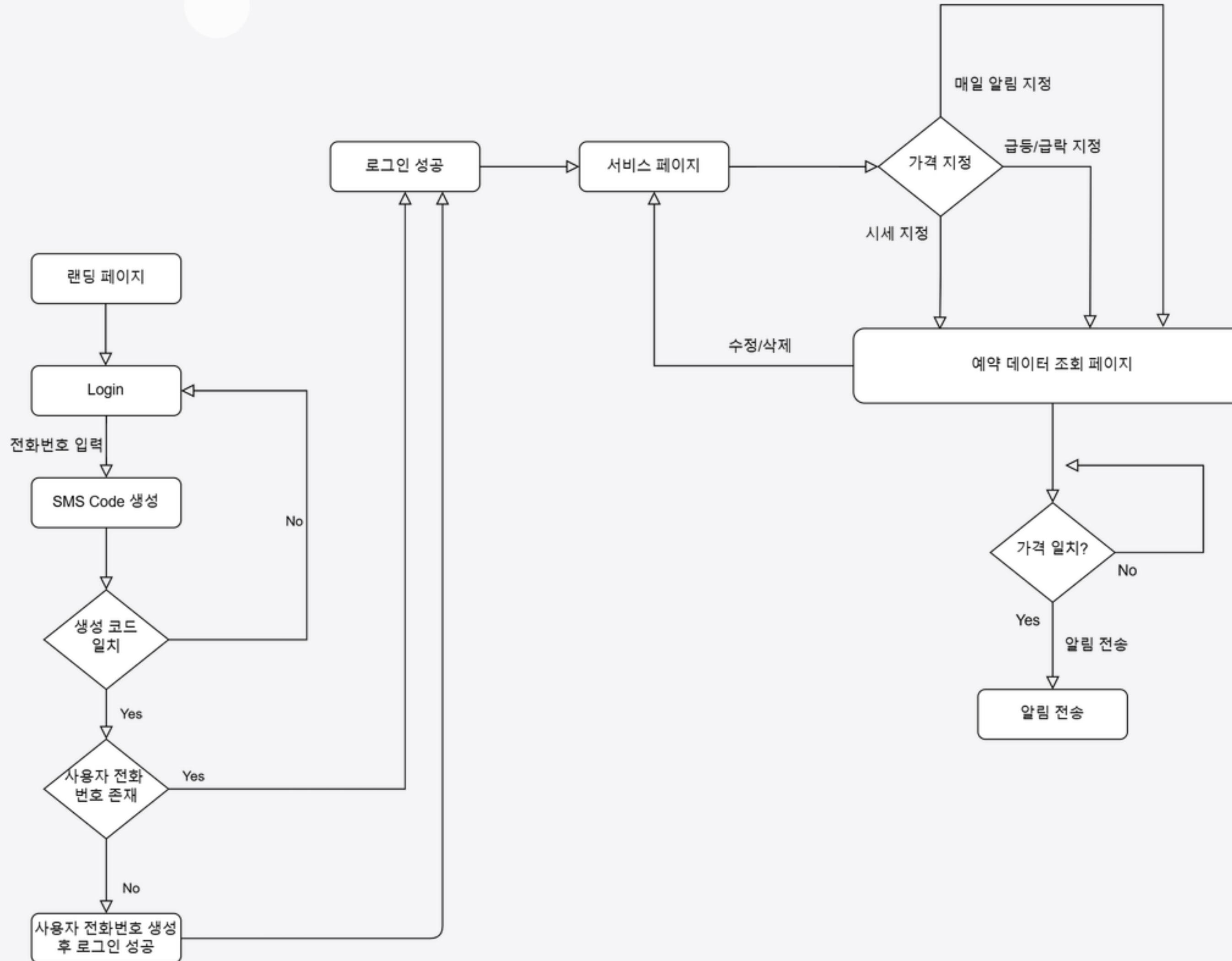
멀티 팩터 인증(MFA) (1)		
유형	식별자	인증
가상	arn:aws:iam::905418305225:mfa/Yoonseok_MFA	해당 사항 없음
페스키 및 보안 키	arn:aws:iam::905418305225:u2f/user/parkyongwoo/58-HEY7GADBERH2DFC552BLKLEY04	0



**Root, IAM 계정 분리 및 권한 최소화  
MFA 멀티 팩터 인증으로 계정 보안 강화**

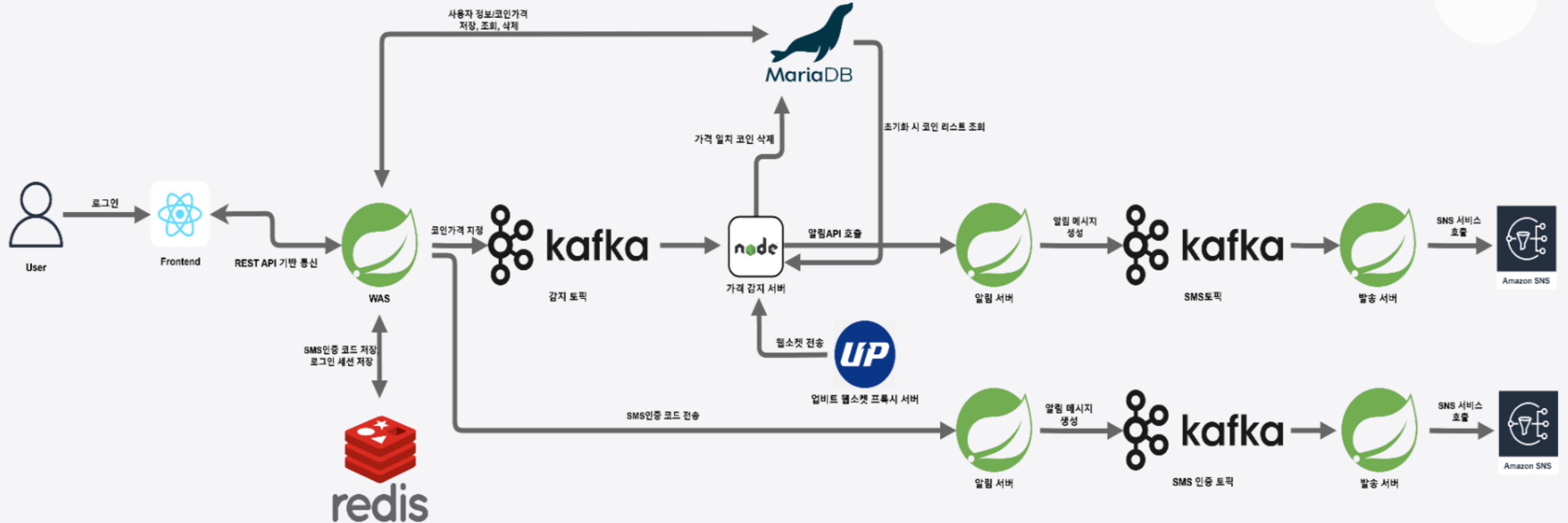
## 02. 개발

### - 플로우 차트



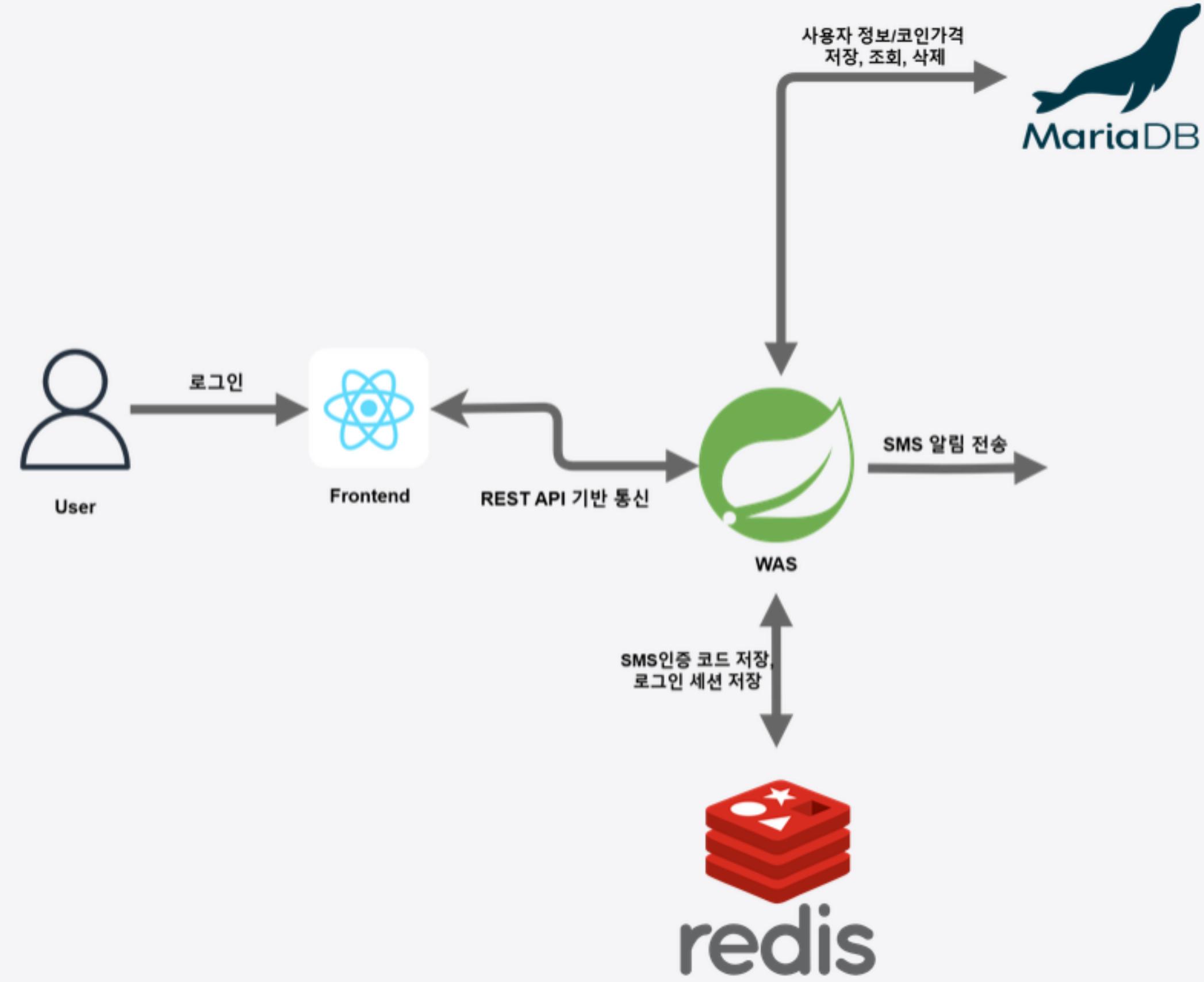
## 02. 개발

### -서비스 플로우



## 02. 개발

### -서비스 플로우



### Front-end

회원은 전화번호 인증으로 로그인을 하고 실시간으로 코인들의 시세, 그래프 추이를 보고 사용자가 원하는 코인에 대한 알림을 추가,수정,삭제를 할 수있음

### WAS

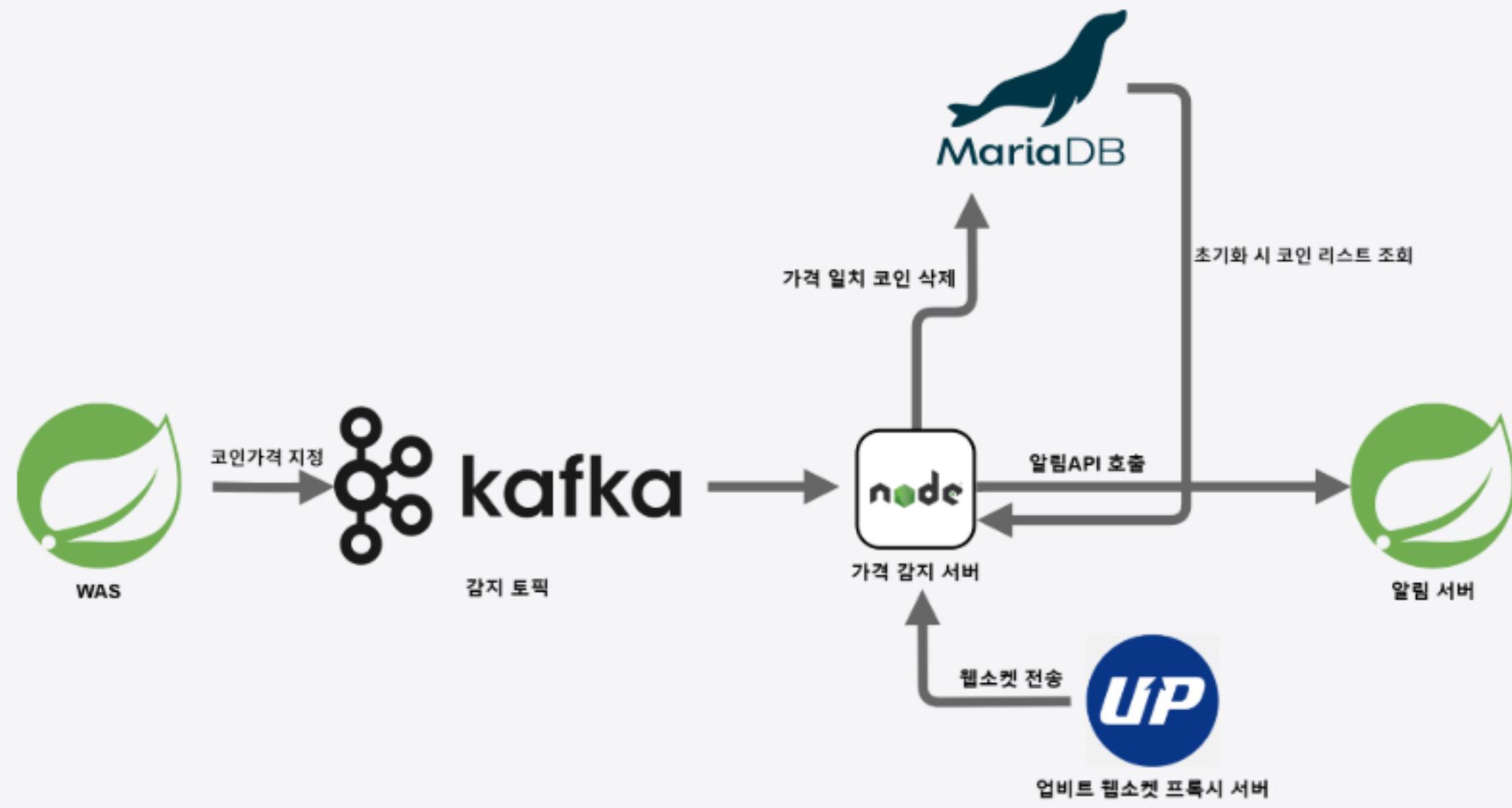
Redis와 mariadb를 이용한 세션 로그인/회원가입  
기능별 알림 추가,수정,삭제,조회 후 카프카 모니터링 토픽을 통해 가격  
감지 서버로 전송

### Redis

로그인/회원가입을 위한 인증코드 발급을 위해 사용자의 전화번호,로그  
인 세션을 저장하는 In-memory DB

## 02. 개발

### -서비스 플로우



### 가격감지 서버

업비트api를 웹소켓 방식으로 사용해 실시간으로 시세를 감지하여 DB의 데이터를 받아와 비교  
예약시 카프카를 통해서 메세지를 받아서 조건을 비교해 조건이 맞다면 알림서버로 전달

### KAFKA

WAS와 가격감지 서버, 알림서버와 발송 서버를 연결하는 메세지 브로커  
알림 메세지를 일시 저장해 시스템이 다운되거나 컨슈머가 다운 시에도  
메세지 전송 가능

### DB

회원 정보 및 예약 코인 정보 저장

## 02. 개발

### -서비스 플로우



### 알림서버

가격감지 서버와 WAS에서 알림 생성 요청을 받아 타입에 맞는 알림메세지 생성 후 카프카 토픽을 통해 발송 서버로 전송



### 알림 발송 서버

알림서버에서 발송한 알림 메세지를 카프카 토픽을 통해 컨슘한 후 서드파티 서비스를 이용해 사용자에게 알림 전송

# “ 하이브리드 클라우드를 이용한 메세지 발송 이중화 ”



이중화

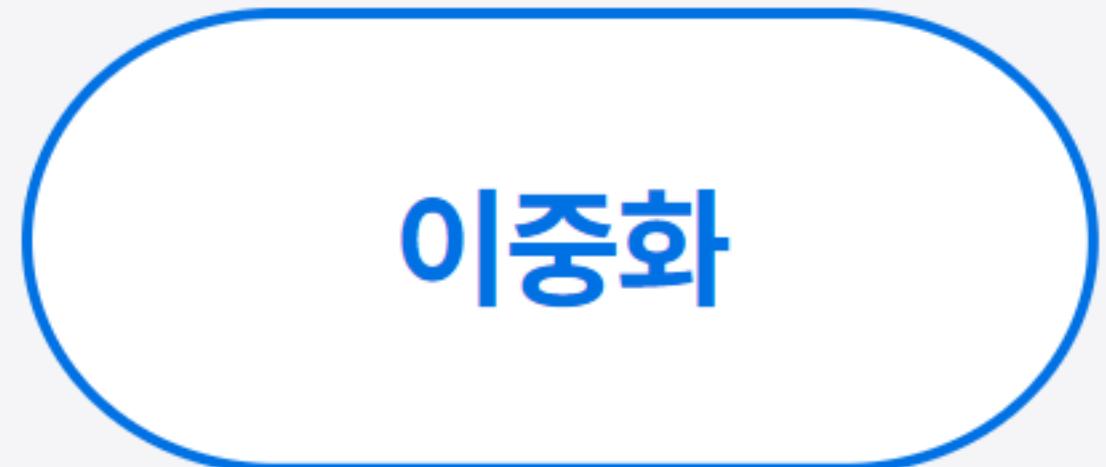


메세지 발송

어디서부터 어디까지?

어떤 메세지를 어떻게?

## 02. 개발



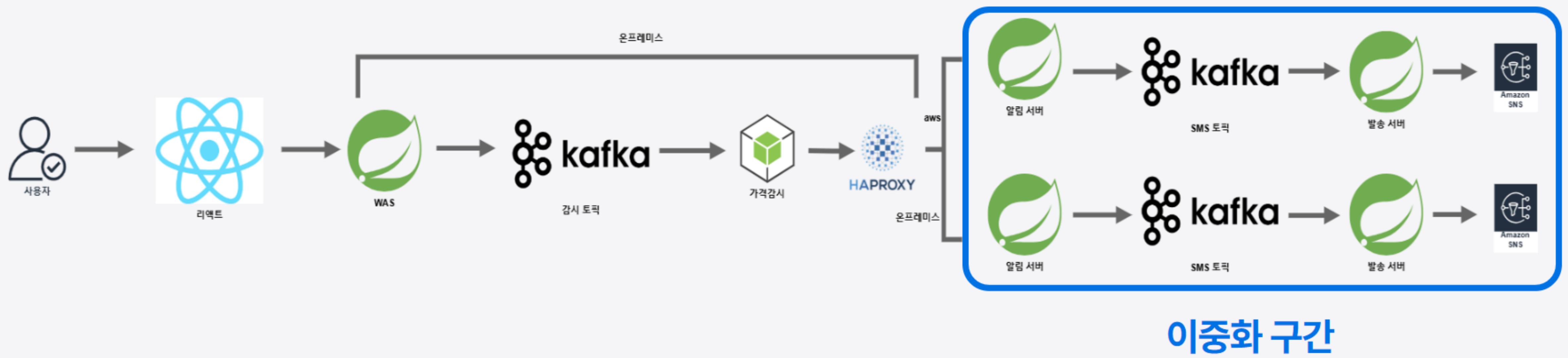
관리 포인트를 줄이고, 메세지 발송을 위한 부분만 이중화  
코인은 변동률이 높으므로 알림이 몰릴 수 있기에 알림 서버  
부터 이중화



사용자가 가장 먼저 반응 할 수 있는 플랫폼인 SMS  
를 선택

## 02. 개발

### - 이중화 다이어그램



## 02. 개발

### - 이중화

### 온프레미스 & 퍼블릭 클라우드 연결



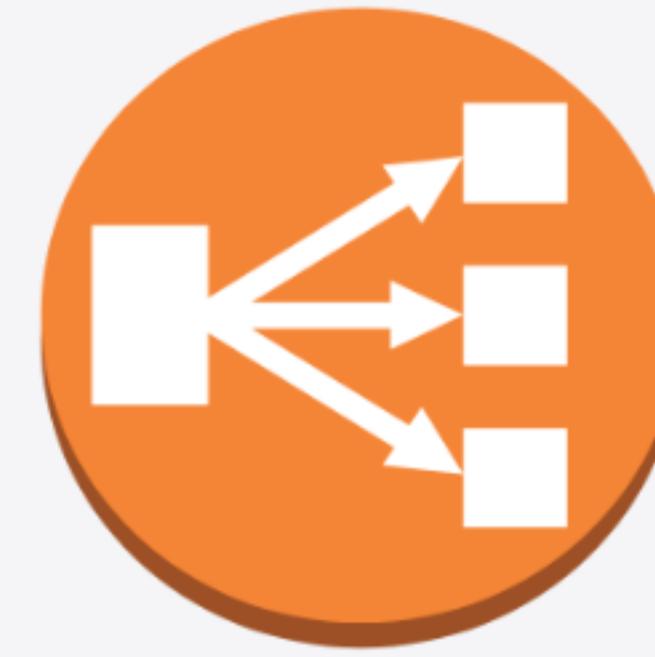
Site to Site VPN

상대적으로 높음

암호화 및 터널링 오버헤드

인터넷 회선 품질 및 장애 영향 가능

VS



Network Load Balancer

매우 낮음 (ms 단위)

고성능, L4 TCP/UDP 기반 로드 밸런싱

AWS 인프라 내부에서 매우 안정적

## 02. 개발

- 이중화

### 온프레미스 & 퍼블릭 클라우드 연결

**실시간 알림을 보내는 서비스의 특성상 자연  
시간이 적은 Network Load Balancer 선택**

상대적으로 높음

암호화 및 터널링 오버헤드

인터넷 회선 품질 및 장애 영향 가능

지연시간

성능성

연결 안정

매우 낮음 (ms 단위)

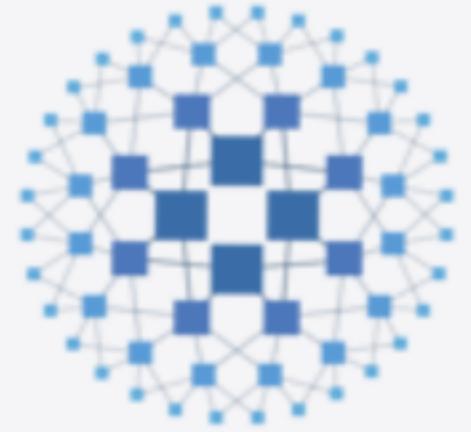
고성능, L4 TCP/UDP 기반 로드 밸런싱

AWS 인프라 내부에서 매우 안정적

## 02. 개발

- 이중화

### 온프레미스 & 퍼블릭 클라우드 연결



**HAPROXY**

HAProxy

고성능 로드밸런서  
상대적으로 단순 (텍스트 기반 설정)

VS



**NGINX**

역할  
설정 난이도

로드밸런서 + 웹 서버 기능  
다소 복잡 (블록 구조 설정)

## 02. 개발

- 이중화

### 온프레미스 & 퍼블릭 클라우드 연결

#### 고성능 부하분산을 위해 HAProxy 선택

고성능 로드밸런서

상대적으로 단순 (텍스트 기반 설정)

역할

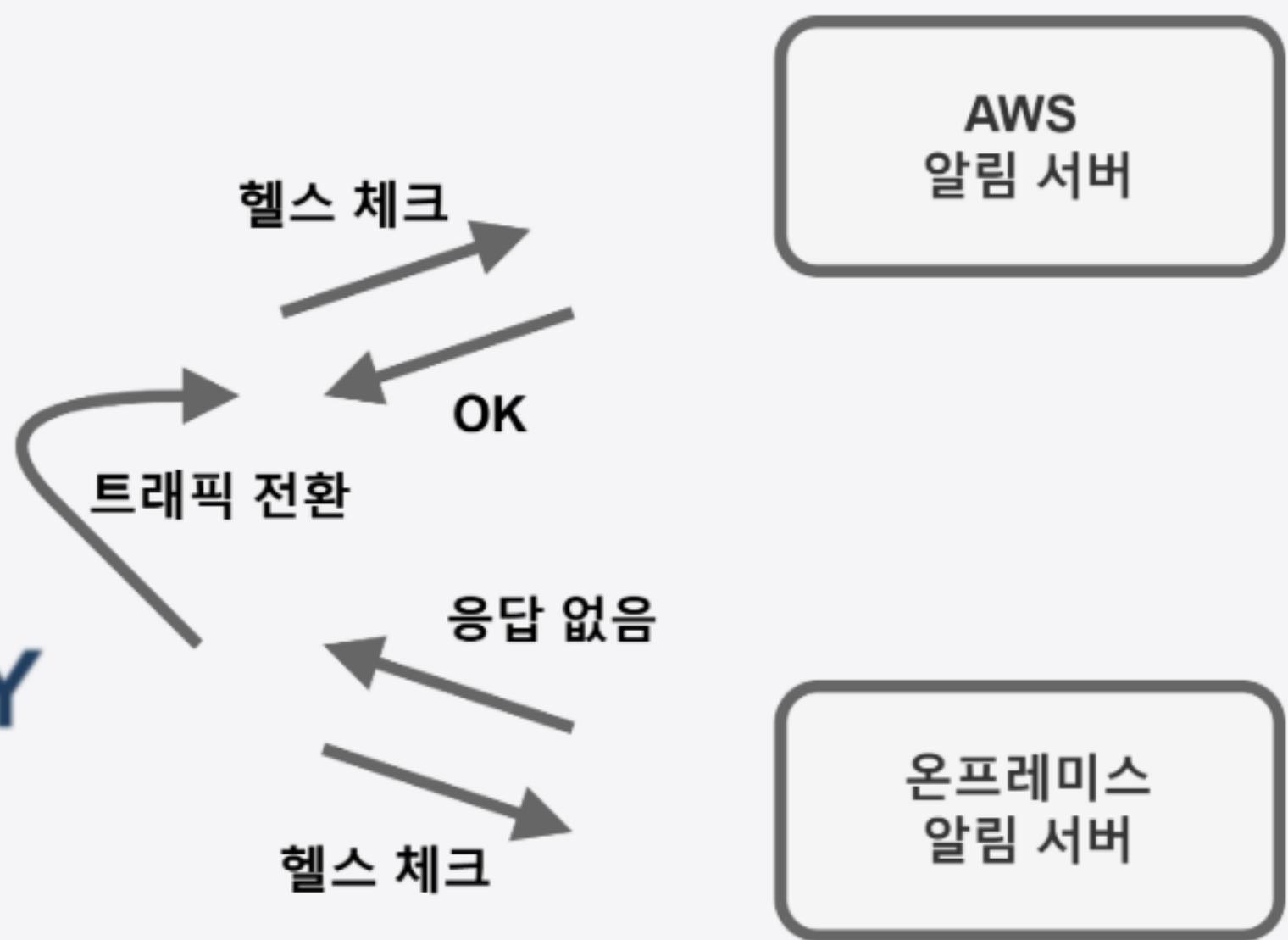
설정 난이도

로드밸런서 + 웹 서버 기능

다소 복잡 (블록 구조 설정)

## 02. 개발

### - Haproxy



- 2초 내에 응답 받지 못하면 비정상 처리
- 온프레미스, aws 중 하나가 응답 하지 않으면 다른 쪽으로 트래픽 전환
- Round Robin 알고리즘을 사용해서 부하를 분산시켜 한쪽은 퍼블릭 클라우드에, 한쪽은 온프레미스상의 알림서버에 가는 Active-Active 구조

## 02. 개발

- On-premise MQ



VS



RabbitMQ

디스크 저장으로 데이터 유실 방지  
초당 수십만 건  
파티셔닝을 통해 쉽게 확장 가능

안정성  
처리량  
확장성

메시지 큐 방식으로 유실 가능성 존재  
초당 수천 ~ 수만 건  
노드 추가 가능하지만 확장성 제한

## 02. 개발

### - On-premise MQ

# 안정성 및 대용량 데이터 처리를 위해 Kafka 선택

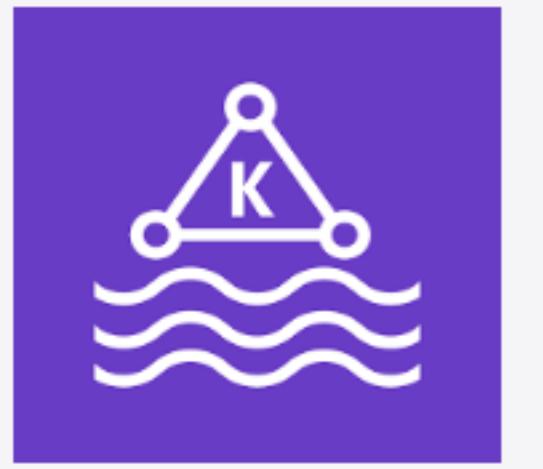
디스크 저장으로 데이터 유실 방지  
초당 수십만 건  
파티셔닝을 통해 쉽게 확장 가능

안정성  
처리량  
확장성

메시지 큐 방식으로 유실 가능성 존재  
초당 수천 ~ 수만 건  
노드 추가 가능하지만 확장성 제한

## 02. 개발

### - Public Cloud (AWS) MQ



**Amazon  
MSK**

**m5.Large**

2 vCPU(고정)  
8GB  
최대 10Gbps

**VS**



**Amazon  
EC2**

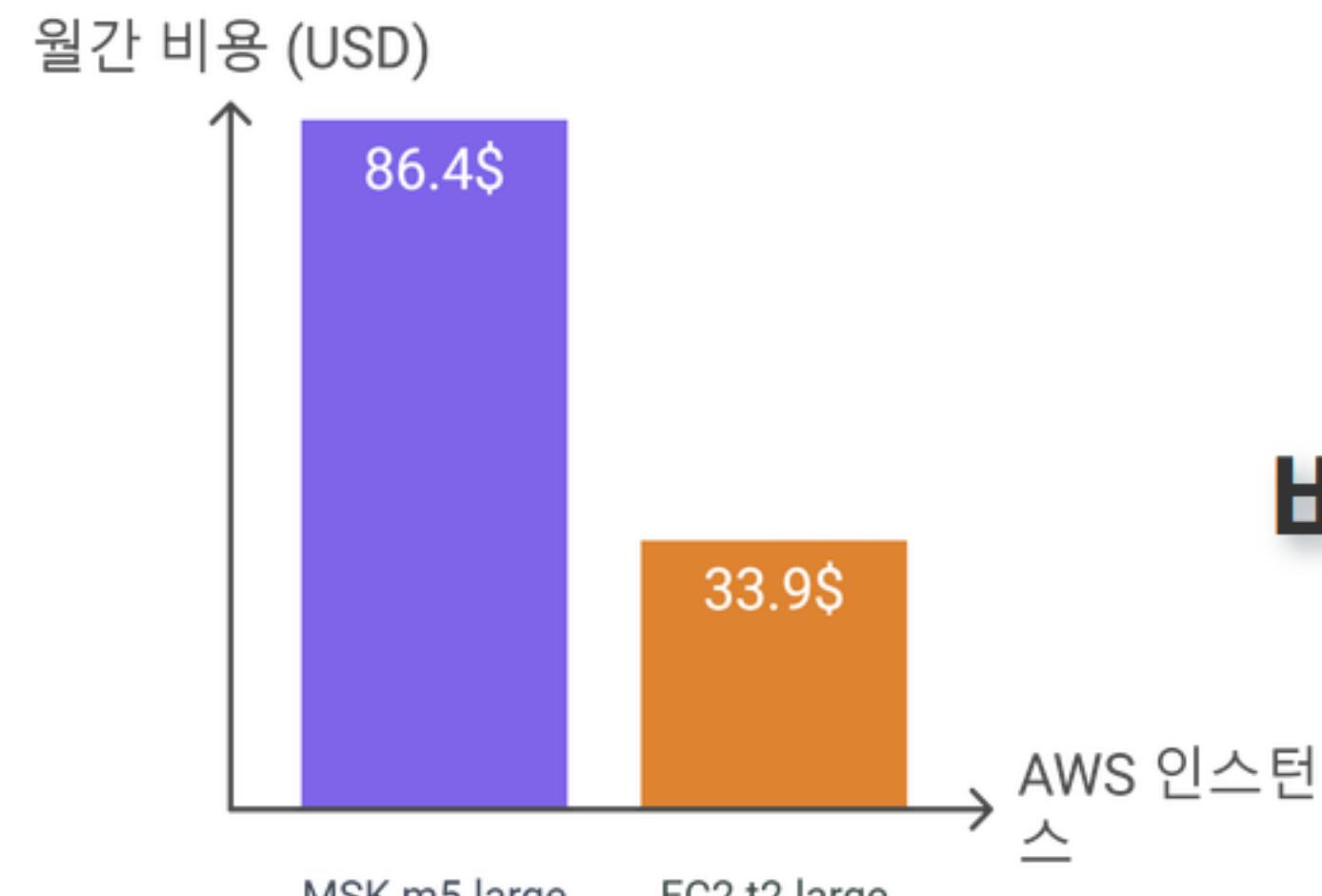
**t2.Large**

2 vCPU(버스트 가능)  
8GB  
최대 5Gbps

**성능**  
**RAM**  
**네트워크 속도**

## 02. 개발

### - Public Cloud (AWS) MQ



AWS 인스턴스 간 비용 비교

**비슷한 성능이지만 약 60%비용 절감**

최대 10Gbps

네트워크 속도

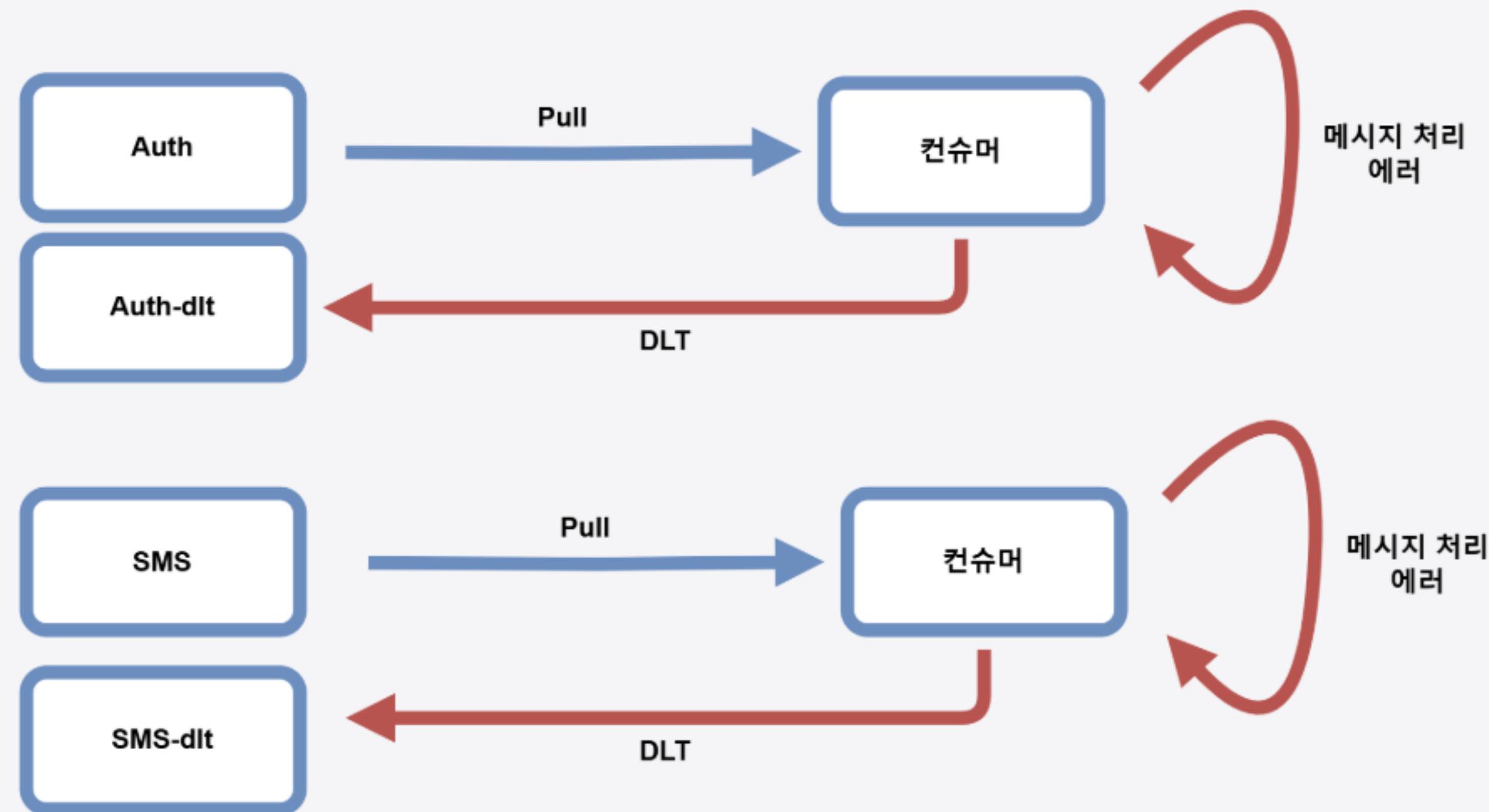
최대 5Gbps

## 02. 개발

### - Kafka

#### Dead Letter Topic(DLT)

**Dead Letter Topic(DLT) : 서비스 문제나 장애로 인해 처리할 수 없는 메시지를 저장하는 별도의 Kafka 토픽**



## 02. 개발

### - 서킷 브레이커 패턴



## 02. 개발

### - 서킷 브레이커 패턴

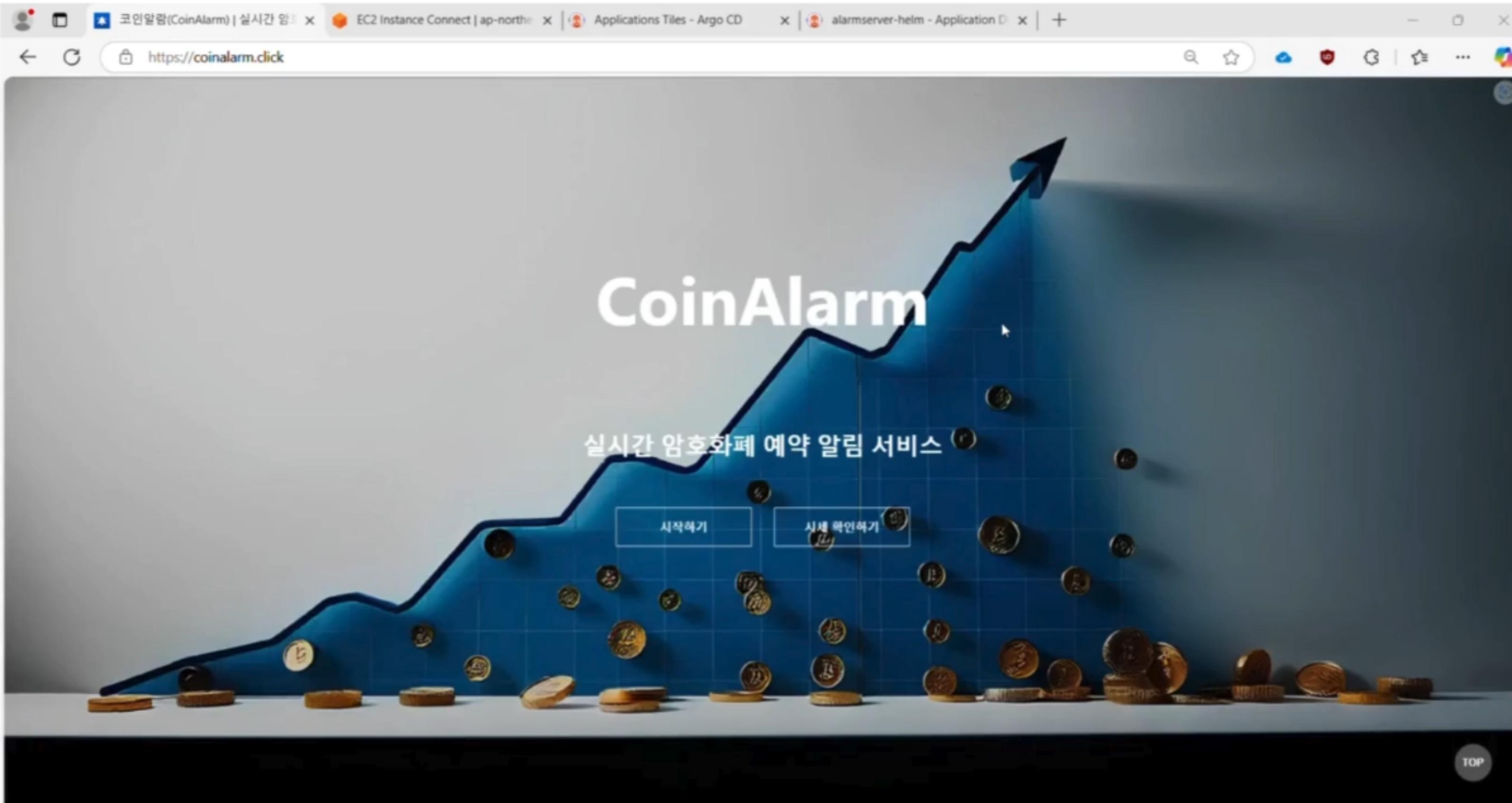
```
2025-02-21 09:27:10 - INFO - o.apache.kafka.clients.NetworkClient - [Producer clientId=producer-1] Node -1 disconnected.  
2025-02-21 09:27:10 - WARN - o.apache.kafka.clients.NetworkClient - [Producer clientId=producer-1] Connection to node -1 (/192.168.56.7:9092) could not be established. Node may not be available.  
2025-02-21 09:27:10 - WARN - o.apache.kafka.clients.NetworkClient - [Producer clientId=producer-1] Bootstrap broker 192.168.56.7:9092 (id: -1 rack: null) disconnected  
2025-02-21 09:27:11 - ERROR - o.s.k.s.LoggingProducerListener - Exception thrown when sending a message with key='AUTH' and payload='{code=446231, phoneNumber=01098047273, message=인증코드[446231], t org.apache.kafka.common.errors.TimeoutException: Topic auth not present in metadata after 60000 ms.  
2025-02-21 09:27:11 - ERROR - c.g.d.a.KafkaProducerService - Fallback executed due to errorSend failed  
2025-02-21 09:27:11 - INFO - c.g.d.a.KafkaProducerService - Circuit Breaker POST /auth  
2025-02-21 09:27:11 - INFO - c.g.d.alarmserver.AlarmController - Sent auth message: 인증코드[446231]  
2025-02-21 09:27:11 - INFO - c.g.d.alarmserver.WebClientService - Fallback request to //auth successful: null  
2025-02-21 09:27:16 - INFO - o.apache.kafka.clients.NetworkClient - [Producer clientId=producer-1] Node -1 disconnected.  
2025-02-21 09:27:16 - WARN - o.apache.kafka.clients.NetworkClient - [Producer clientId=producer-1] Connection to node -1 (/192.168.56.7:9092) could not be established. Node may not be available.  
2025-02-21 09:27:16 - WARN - o.apache.kafka.clients.NetworkClient - [Producer clientId=producer-1] Bootstrap broker 192.168.56.7:9092 (id: -1 rack: null) disconnected
```

알림서버에서 알림발송 서버로 가는 카프카가 다운이 됐을 시 AWS에 있는 카프카로 전달이 되도록 구현

사용자가 SMS 로그인 시 서킷브레이커 패턴으로 온프레미스 카프카가 다운되었을 때 AWS에 구현이 되어있는 알림서버로  
인증코드 전달

## 02. 개발

### 시연영상



랜딩페이지 진입 -> 서비스 소개 및 실시간 코인 시세 조회

# 목차

01. 서비스 ↗

02. 개발 ↗

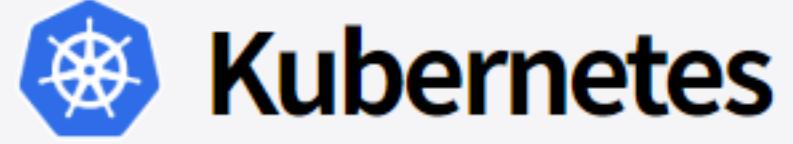
03. 인프라 ↗

04. 모니터링&테스트 ↗

05. 회고 ↗

06. Q&A ↗

### 03. 인프라



Kubernetes

컨테이너 오케스트레이션

트래픽 로드밸런싱

무중단 배포 및 롤백

유연한 컨테이너 스케일 아웃

## 03. 인프라

### - K8s 시방서

#### Kubernetes Cluster 설계

##### 클러스터 구성

항목	설명
Server	CPU : Intel i5-14400 RAM : 64GB
Kubernetes 버전	1.30
컨트롤 플레인	1대 (VirtualBox VM)
워커 노드	2대 (VirtualBox VM)
네트워킹 플러그인	Calico

##### Service

Type	Name	Labels	Namespace	targetPort
ClusterIP	alarmserver-svc	app: alarmserver	dev	8090
ClusterIP	auth_workerserver-svc	app: auth_workerserver	dev	8091
ClusterIP	price_workerserver-svc	app: price_workerserver-svc	dev	8092
LoadBalancer	was-svc	app: was	dev	8080

#### VM 구성 및 역할

VM 역할	IP 주소	설명
Kubernetes 컨트롤 플레인 노드	192.168.56.5	CPU : 2 core Memory : 6GB
Kubernetes 워커 노드 1	192.168.56.6	CPU : 2 core Memory : 4GB

#### 스토리지 및 네트워킹

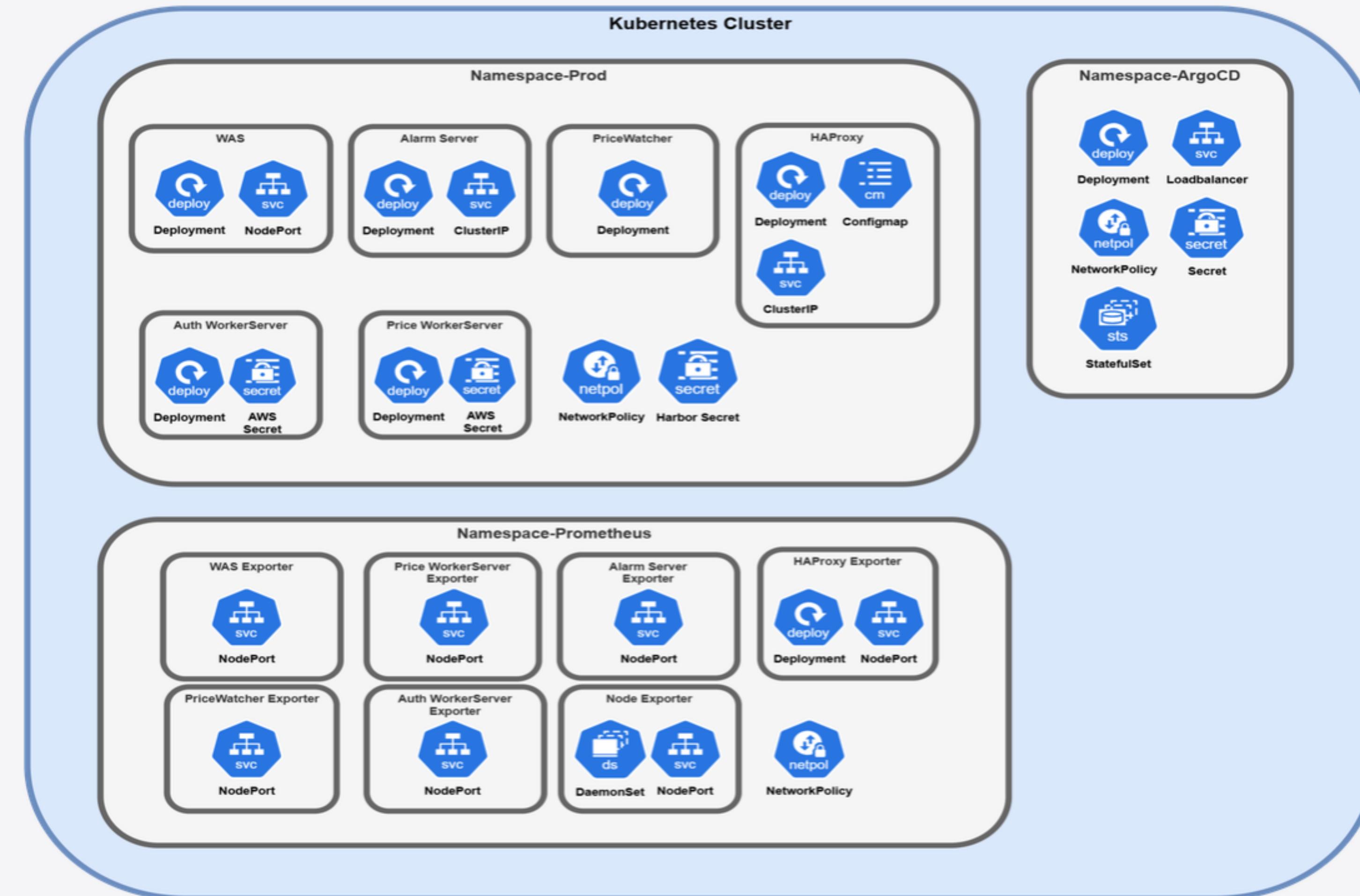
항목	설명
네트워킹	Calico로 네트워크 구성

설정 값	설명
	Kubernetes 네트워킹
/12	Kubernetes 파드 네트워크 CIDR

## 03. 인프라

### - On-Premise

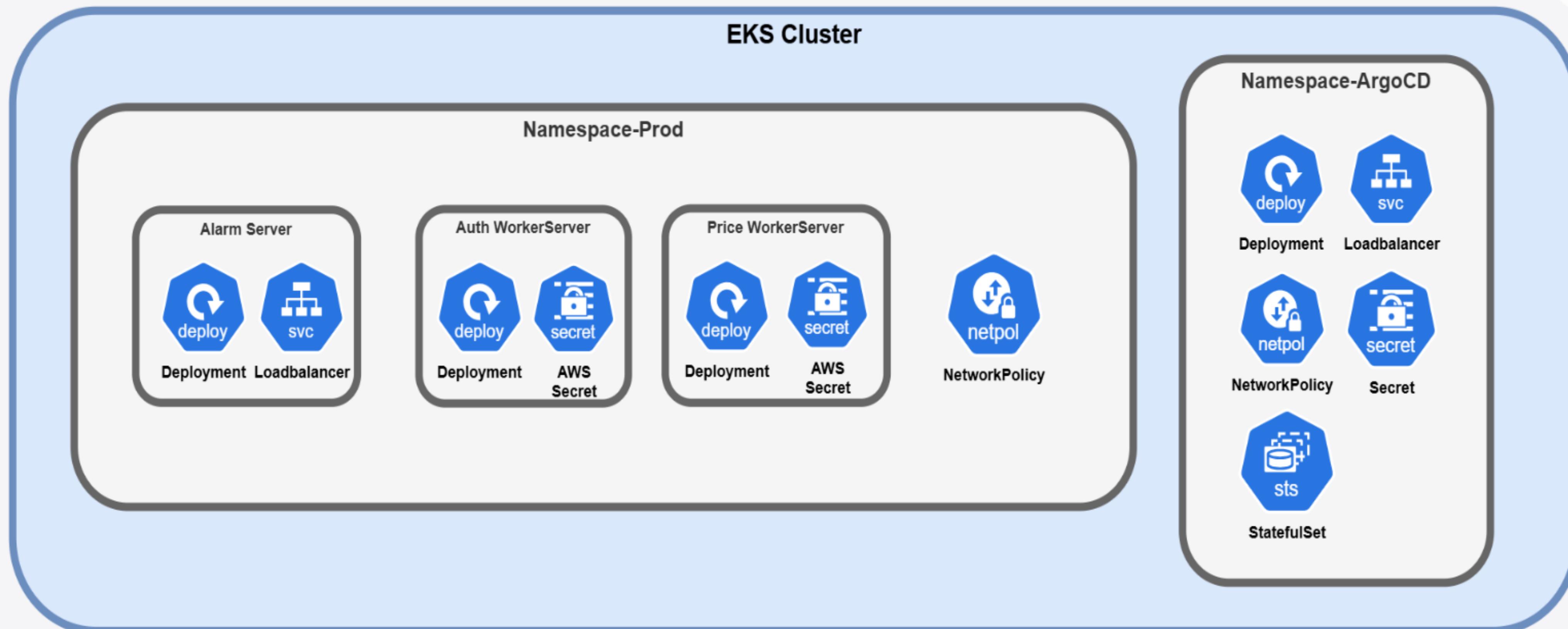
#### 쿠버네티스 클러스터 다이어그램



## 03. 인프라

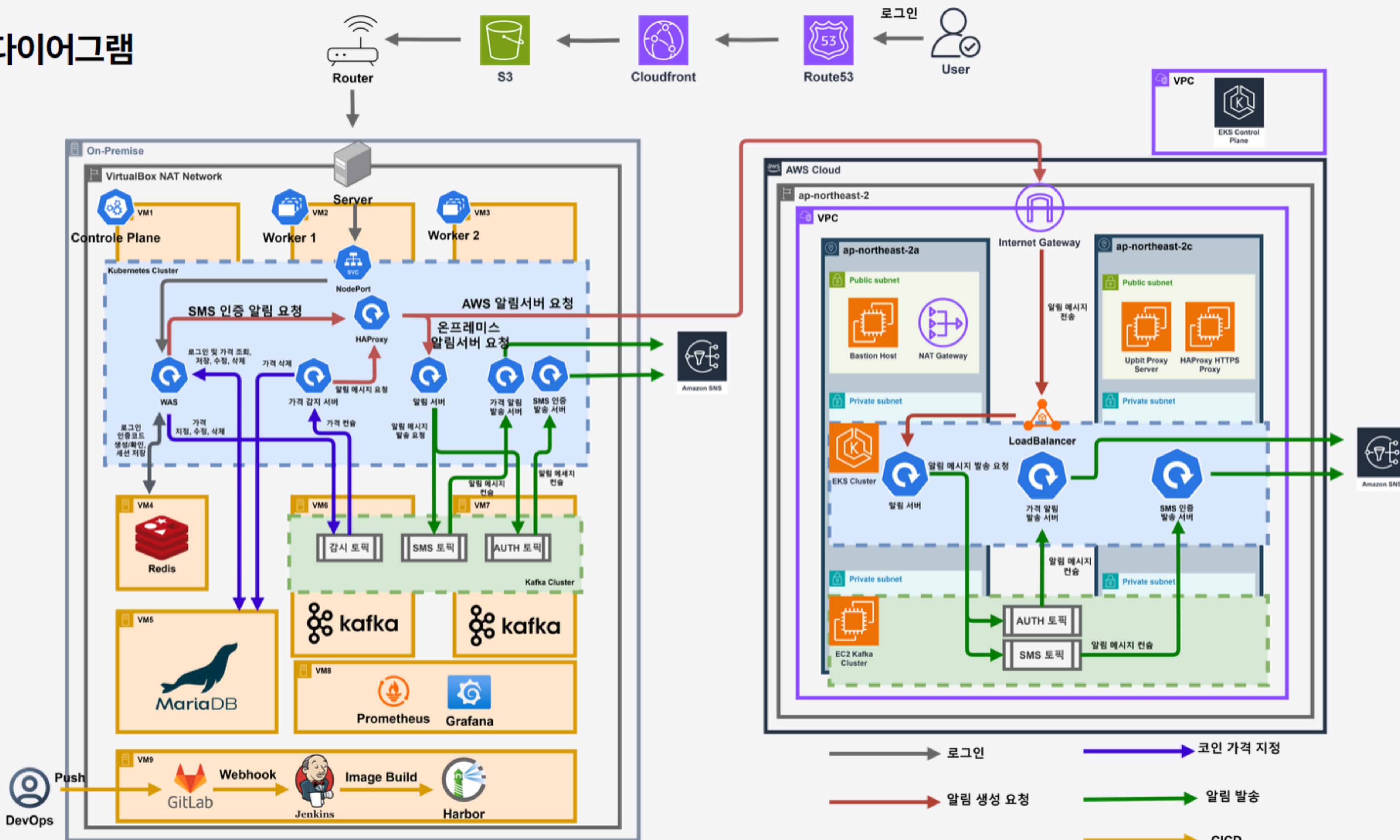
### - Public Cloud (AWS)

EKS 다이어그램



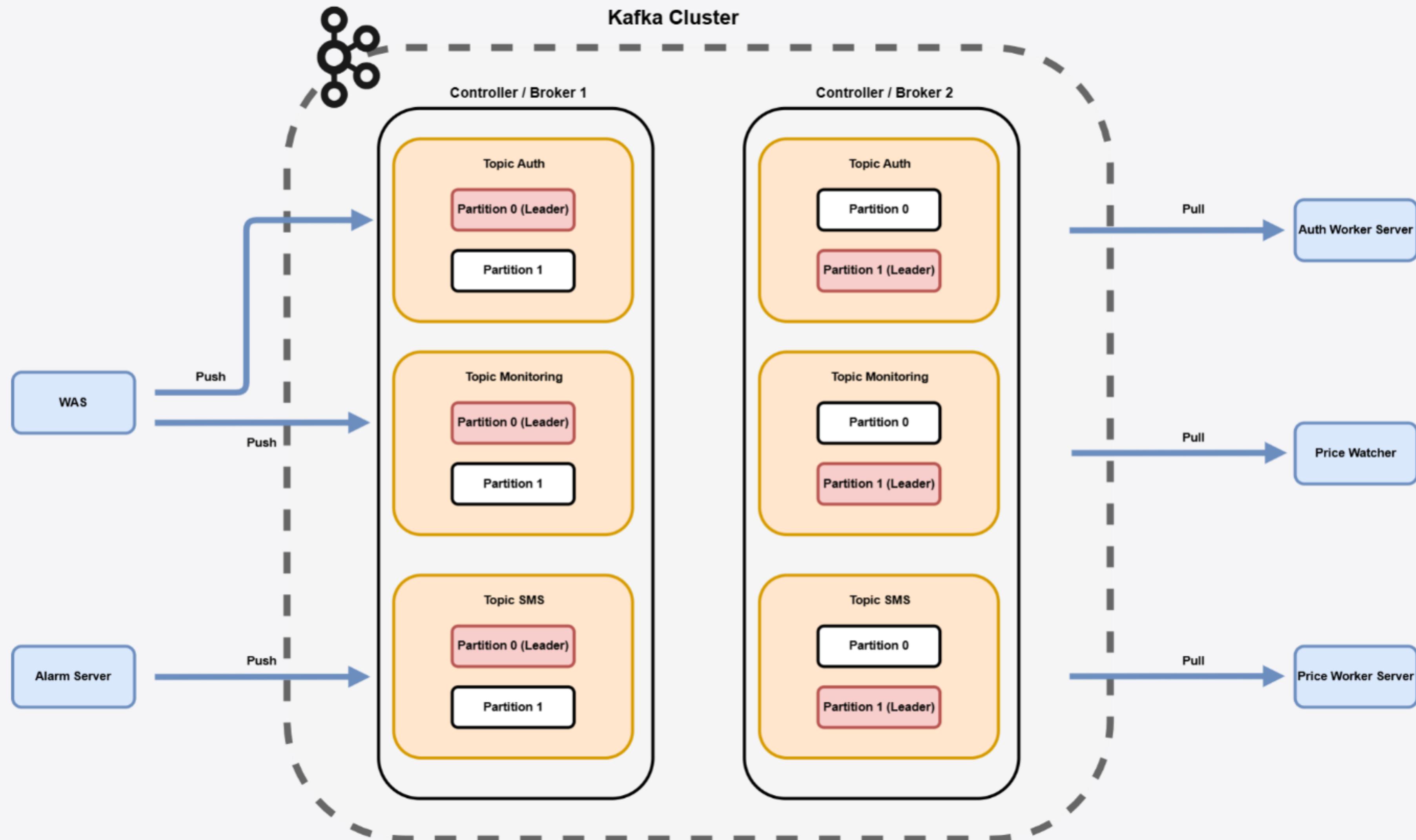
## 03. 인프라

### - 인프라 다이어그램



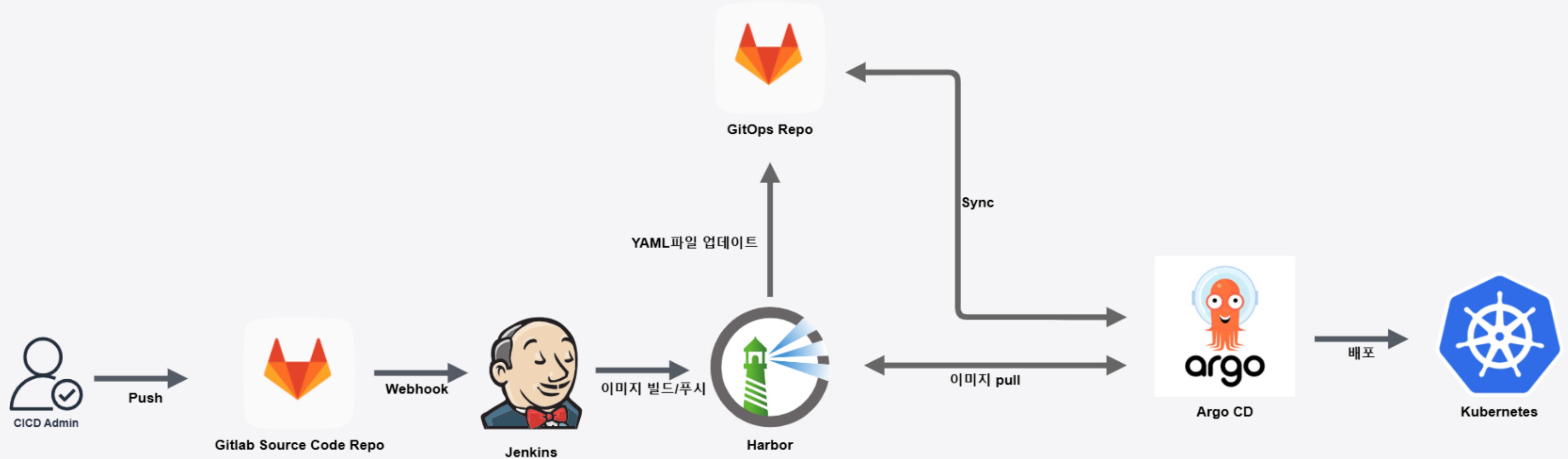
## 03. 인프라

### - Kafka 클러스터 다이어그램



## 03. 인프라

### - CI/CD



## 03. 인프라

### - CI/CD 영상

The screenshot shows the IntelliJ IDEA interface. The left sidebar displays a project structure with files like KafkaProducerConfig.java, application.yaml, AlarmController.java, KafkaProducerService.java, and MetricConfig.java. The main editor window shows the content of application.yaml. The terminal window at the bottom shows the following git commands being run:

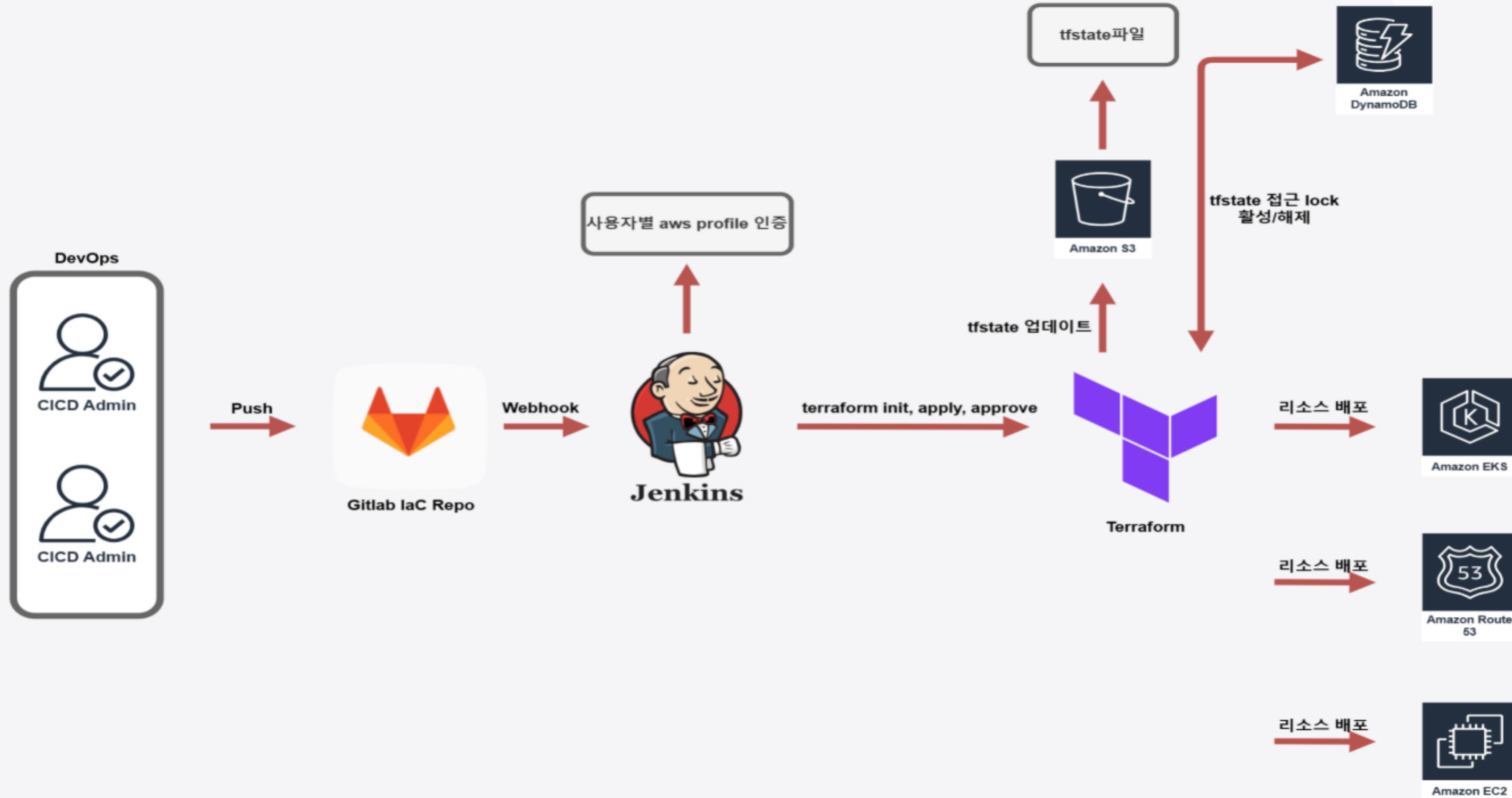
```
Compressing objects: 100% (6/6), done.
Writing objects: 100% (10/10), 677 bytes | 677.00 KiB/s, done.
Total 10 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
To http://gitlab.dragonhailstone.org/autoever_message/alarmserver.git
  700c773..5405314  dev -> dev
PS C:\Users\yongw\OneDrive\바탕 화면\autoever_message\alarmserver> git add .
PS C:\Users\yongw\OneDrive\바탕 화면\autoever_message\alarmserver> git commit -m "test"
[dev 8d3bd79] test
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\yongw\OneDrive\바탕 화면\autoever_message\alarmserver>
```

A tooltip in the bottom right corner of the terminal area says: "IntelliJ IDEA 평가판이 만료되었습니다. IntelliJ IDEA을(를) 계속 사용하려면 25.2.21의 유료한 구독이 필요합니다." (IntelliJ IDEA evaluation has expired. To continue using IntelliJ IDEA, you need to purchase a valid subscription for version 25.2.21.)

**GITLAB에 PUSH**

## 03. 인프라

### - IaC



# 목차

01. 서비스 ↗

02. 개발 ↗

03. 인프라 ↗

04. 모니터링&테스트 ↗

05. 회고 ↗

06. Q&A ↗

## 04. 모니터링 & 테스트

### - Prometheus



..... **node\_exporter**: 노드 상태 및 자원 사용량



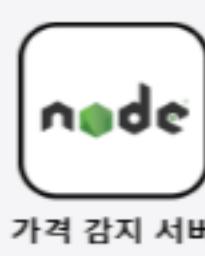
..... **Actuator + Micrometer**

Actuator는 애플리케이션의 상태를 확인  
Micrometer는 메트릭을 Prometheus에 전송



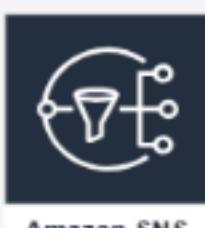
..... **Kafka\_exporter**

kafka 클러스터의 메트릭을 수집하기 위한 도구



..... **prom-client 라이브러리**

Node.js 애플리케이션에서 Prometheus 메트릭을  
수집하는 라이브러리



..... **CloudWatch 서비스**

CloudWatch를 통해 AWS SNS 서비스의 모니터링 및  
로그 관리

## 04. 모니터링 & 테스트

### - Prometheus



WAS, 알림서버, 발송 서버

CPU 사용량, HTTP 요청 소요 시간, 메모리  
사용량, HEALTH 체크, 총 요청 수



카프카 클러스터

총 소비된 메세지 개수, 보내지 못한 메세지 개  
수, 카프카 토픽, 클러스터 상태



가격 감지 서버

HTTP 요청 소요 시간, HEALTH 체크,  
CPU 사용량, 메모리 사용량



쿠버네티스 노드

노드의 상태, CPU 사용량, 메모리 사용량, 네트  
워크 트래픽

## 04. 모니터링 & 테스트

### - Grafana



## 04. 모니터링 & 테스트

### - 부하 테스트

#### K6 vs JMeter

스크립트 작성 용이함을 위해 JAVA, JAVASCRIPT 중에서 선정  
간단한 HTTP 요청을 테스트 하기 위해 비교적 간단한 K6 를 선정

## 04. 모니터링 & 테스트

### - 부하 테스트

#### - 테스트 시나리오

### K6 성능 테스트

성능 테스트 +				☰ ↕ ⚡ 🔍
Aa 구간	⌚ 테스트 유형	⌚ 타입	☰ 세부내용	🕒 진행상황
📄 WAS	Load Test	POST	10분간 초당 10~500 요청을 처리, 95%의 성공률	🟢 완료
WAS	Stress Test	POST	WAS가 보낼 수 있는 최대 요청 수 임계값 측정	🔵 진행 중
📄 WAS	Spike Test	POST	1분간 초당 50개의 요청을 처리 후 10초간 1000개 요청을 발생, 요청 처리 실패율 측정	🔵 진행 중
가격 감지	Stress Test	JSON	가격 감지 서버가 데이터를 받아서 보낼 수 있는 최대 개수 테스트	🔵 진행 중
알림서버	Load Test	POST	10분간 100~1000명의 POST요청을 초당 10~100개씩 처리, 95%의 성공률	🔵 진행 중
알림서버	Stress Test	POST	알림 서버가 처리할 수 있는 최대 요청 개수를 테스트	🔵 진행 중
📄 알림서버	Soak Test	POST	3시간 동안 초당 100개의 요청을 지속적으로 발생	🔵 진행 중
📄 발송서버	Load Test	JSON	초당 100~500개의 메시지를 카프카에서 받아 초당 50~100개의 SMS 전송, 성공률 98%	● 시작 전
발송서버	Soak Test	JSON	1시간 동안 초당 100개의 SMS를 지속적으로 전송	● 시작 전

## 04. 모니터링 & 테스트

### - 부하 테스트

#### - Load Test

10분간 초당 10~500 요청을 처리, 95%의 성공률 목표

```
✓ ✅ Status is 201
✗ ✅ Response time < 500ms
↳ 99% - ✓ 7347 / X 2

checks.....: 99.98% 14696 out of 14698
data_received.....: 1.4 MB 2.3 kB/s
data_sent.....: 858 kB 1.4 kB/s
http_req_blocked.....: avg=24.19µs min=198ns med=379ns max=174.44ms p(90)=510ns p(95)=577ns
http_req_connecting.....: avg=1.53µs min=0s med=0s max=11.27ms p(90)=0s p(95)=0s
✓ http_req_duration.....: avg=73.87ms min=44.57ms med=67.39ms max=3.78s p(90)=96.64ms p(95)=111.78ms
{ expected_response:true }.....: avg=73.87ms min=44.57ms med=67.39ms max=3.78s p(90)=96.64ms p(95)=111.78ms
✓ http_req_failed.....: 0.00% 0 out of 7349
http_req_receiving.....: avg=232.73µs min=24.25µs med=188.72µs max=11.24ms p(90)=390.28µs p(95)=718.92µs
http_req_sending.....: avg=2.08ms min=51.61µs med=884.5µs max=3.78s p(90)=3.89ms p(95)=4.7ms
http_req_tls_handshaking.....: avg=2.09µs min=0s med=0s max=15.38ms p(90)=0s p(95)=0s
http_req_waiting.....: avg=71.54ms min=32.56µs med=65.64ms max=641.14ms p(90)=94.57ms p(95)=109.96ms
http_reqs.....: 7349 12.247035/s
iteration_duration.....: avg=81.58ms min=49.51ms med=74.97ms max=3.79s p(90)=105.38ms p(95)=120.43ms
iterations.....: 7349 12.247035/s
vus.....: 1 min=1 max=1
vus_max.....: 1 min=1 max=1

running (10m00.1s), 0/1 VUs, 7349 complete and 0 interrupted iterations
default ✓ [=====] 1 VUs 10m0s
```

#### 성공률

- 99.98%

(2개의 요청만이 500ms를 넘어감)

- 실패한 요청 없음

#### 응답시간

- 평균 응답 시간은 73.87ms

- 90~95%의 요청은 100ms 내외로 완료됨

- 일부 요청(2개)이 500ms를 초과

#### 처리량

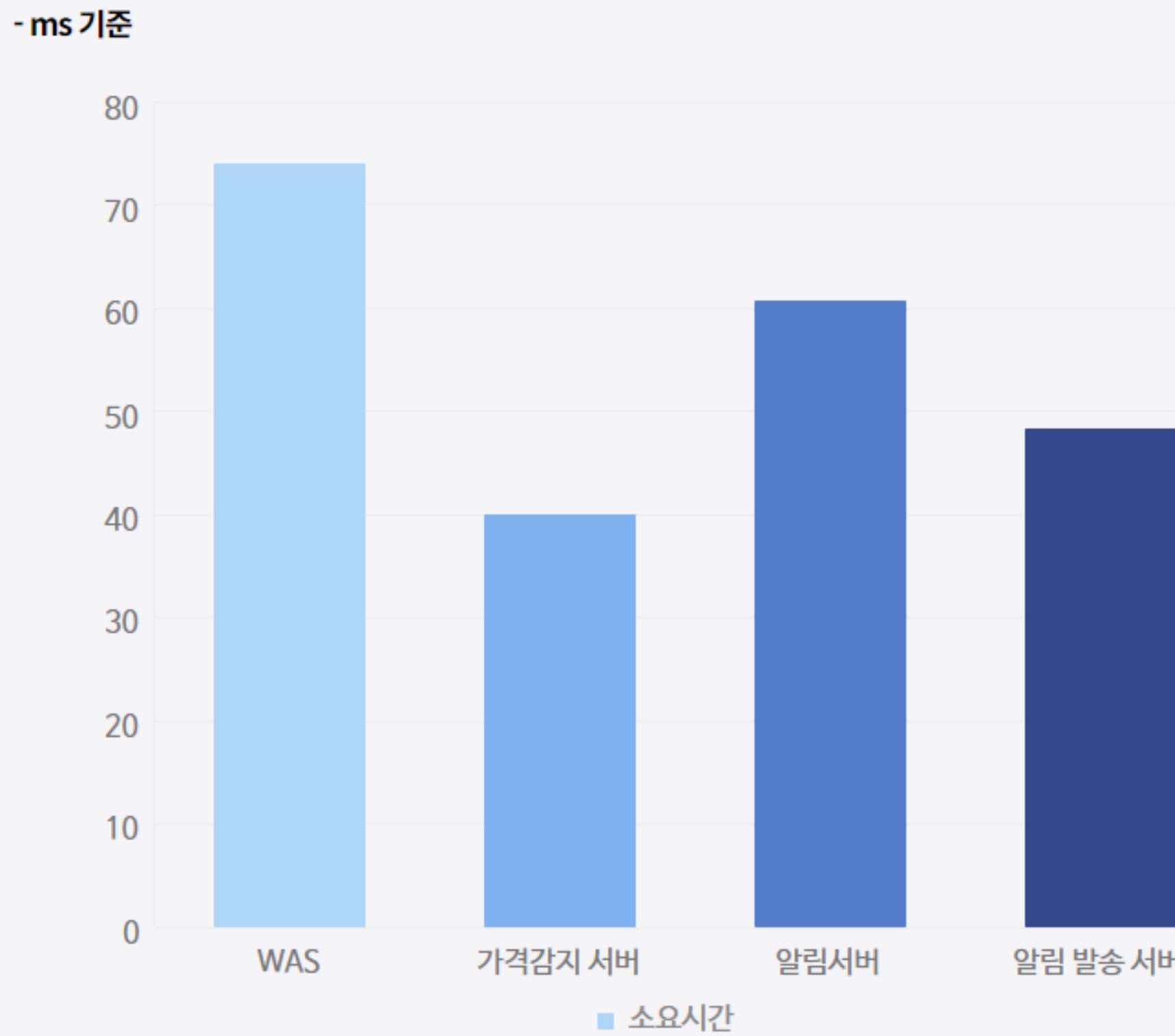
- 10분 동안 총 7349개 요청 수행

## 04. 모니터링 & 테스트

### - 부하 테스트 결과

#### - Load Test

10분간 초당 10~500 요청을 처리, 95%의 성공률



## 요청 처리 시간

WAS - 73.87ms

가격감지 서버 - 55.5 ms

알림서버 - 60.7ms

알림 발송 서버 - 48.2ms

카프카 프로듀서 - 10000개 기준 62.82 ms

카프카 컨슈머 - 10000개 기준 280.17 ms

### 총 요청 처리 시간

$73.87\text{ms} + 62.82\text{ms} + 55.5\text{ms} + 60.7\text{ms} + 280.17\text{ms}$

$+ 48.2\text{ms} \Rightarrow 581.26 \text{ ms}$

# 목차

01. 서비스 ↗

02. 개발 ↗

03. 인프라 ↗

04. 모니터링&테스트 ↗

05. 회고 ↗

06. Q&A ↗

## 05. 회고



### 박용우

온프레미스 주소에 HTTPS 인증서를 적용하려면 도메인이 필요했지만, 사용할 수 있는 온프레미스 주소 포트가 제한적이어서 EC2에 HAProxy를 설치하고 HTTPS 인증서를 적용한 후 온프레미스 노드 포트로 프록시하여 해결했습니다.

재해 복구 대책을 충분히 마련하지 못한 점이 아쉬웠으며, 특히 Kafka 미러링 등의 대비책을 구체적으로 구현하지 못했습니다.

### 지윤석

배포 과정에서 CloudFront와 Route 53을 연결하여 HTTPS 라우팅을 구성하려 했으나, 잘못된 리전에서 SSL 인증서를 발급받아 오류가 발생하여 us-east-1 리전에서 재발급하였습니다.

테스트 시나리오를 작성하고 테스트를 진행하려 했지만, 기본적인 로드 테스트만 한 점이 아쉬웠습니다. 교육 과정이 끝나고 나면 테스트 시나리오를 기반으로 다양한 테스트를 진행해 보고 싶습니다.



# Q & A