구름대학교 수강신청

TEAM 2조 G3M

정한교, 신현식, 이승엽, 최수환, 배수빈

목차

01 프로젝트 개요

대학교 수강신청 서비스 (구름 대학교 수강신청)

- 학생들이 수강신청 기간에 각 학기마다 수강할 강의를 선택하고 등록할 수 있는 온라인 서비스. 학생들은 이를 이용하여 **강의 검색, 수강 신청, 강의 일정 확인** 등을 한다.

- ***안정적인 인프라를 구축하고, 대규모 트래픽을 처리할 수 있는 서비스를 구현한다.***

01 프로젝트 개요

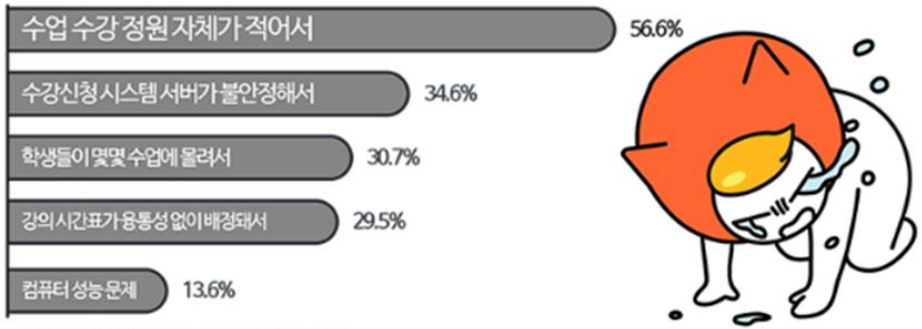
주제 선정 배경

대학교 수강신청 과정에서 발생하는 여러 문제점 중 가장 큰 비중을 차지하는 서버 문제

대학생 10명 중 3명, 2학기 수강신청 '실패'

* 대학생 1,384명 대상 설문조사 결과, 자료제공 : 알바몬

Q. 수강신청 실패 이유는? (*복수응답)



*수강신청에 실패했다고 답한 응답자 대상

■ WISE에서 개선돼야 한다고 생각되는 부분은 무엇입니까? (복수응답 가능)

1	서버 수용 인원 및 접속 대기 시간	73.6%
2	'세션 완료' 관련 오류	72.1%
3	느린 반응 속도	40.2%
4	인터넷 익스플로러만 접속 가능한 점	35.8%
5	기타	8.5%

■ 수강신청에서 개선돼야할 점은 무엇이라고 생각하십니까? (복수응답 가능)

1	다양한 강의 제공	51.5%
2	강의를 수강할 수 있는 전체 수용인원 확대	28.7%
3	대학행정정보시스템(WISE) 개선	11.5%
4	강의에 대한 명확한 사전 정보 제공	6.8%
5	기타	1.5%

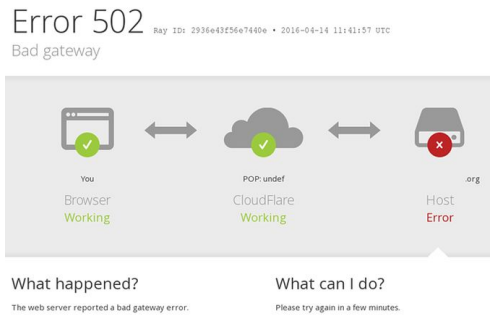
• 설문기간: 3월 6일부터 3월 11일까지 6일간
• 진행: 서울시립대신문 페이스북, 직접 설문
• 응답자 수: 586명

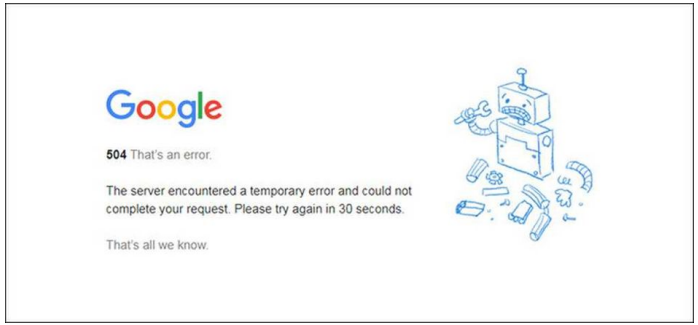
01 프로젝트 개요

주제 선정 배경

수강 신청 기간에 동시에 수강 신청을 하거나, 수업 정보를 조회하는 등의 작업으로 인해 서버 부하 증가 및 서버 지연으로 시스템 자체가 다운되거나 느려지는 등의 문제가 발생

이러한 문제는 학생들의 불편함으로 이어짐





01 프로젝트 개요

주제 선정 이유

G3M 팀은 인프라 구축을 전문으로 하는 팀

G3M 팀은 수강신청 서비스를 안정적이게 서비스를 제공할 수 있는 능력이 있음

분산 시스템을 통해 내구성 있는 인프라를 구축하고, 클라우드 서비스를 이용하여 대규모 트래픽을 처리하도록 하여 원활한 수강 신청 서비스를 제공하게 된다면 여러 문제점을 가진 현행 수강 신청 서비스와 관련된 문제를 해결할 수 있다!

01 프로젝트 개요

AS IS (현황)

- **수업 정보 조회**에 많은 시간이 걸려서 불편함
- 수강 신청이 열리기를 기다리는데 시간을 다른 사이트에서 봐야함
- 학생들이 조회를 위해서 지속적인 새로고침을 하여 서버 조회가 다중 요청되어 서버 지연
- 수강신청 기간에 동시에 수강신청을 하거나, 수업 정보를 조회하는 등의 작업으로 인해 서버 부하증가 및 서버 지연으로 시스템 자체가 다운되거나 느려지는 등의 문제가 발생

TO-BE (개선)

- 다수의 백엔드에서의 로드밸런싱과 데이터베이스 성능 최적화및 캐시 서버를 도입하여 빠른 데이터 접근을 지원함
- 수강신청 페이지 안에서 수강신청 시간까지 남은 시간을 표시하여 사용자의 불편함을 개선
- 반응형 기능으로 새로고침을 하지 않아도 지속적인 서비스 이용이 가능하게 되어 시스템의 부하를 줄임
- 대규모 트래픽을 처리할 수 있는 분산 시스템을 구축하여 수강신청 시스템의 확장성 및 가용성을 높임과 동시에 수강신청 기간, 수강신청 아닌기간에 맞춰 서버를 오토스케일링하여 원활한 서비스이용및 시스템 자원 절약

01 프로젝트 개요

프로젝트 환경 - 서비스

웹 서비스를 DB, Backend, Frontend 로 구성하는 방식으로 개발
DB는 가장 범용적인 관계형 데이터베이스인 MySQL를 사용
Backend로는 실무에서 많이 사용하는 Spring을 사용하며 많은 설정을 간편하게 처리해주는 Spring boot를 사용
Frontend는 재사용을 통해 애플리케이션의 개발 기간을 단축할 수 있는 Vue.js를 사용

01 프로젝트 개요

프로젝트 환경 - 인프라

프로젝트의 목적인 대규모 트래픽 처리를 위해 AWS 환경을 사용
AWS의 많은 서비스들 중에서 안정적인 인프라 구성을 위해 EKS와 RDS를 사용
DNS 서비스로 Route 53을 사용

01 프로젝트 개요

프로젝트 환경 - CICD

앞의 인프라 환경에 CICD의 환경을 구축
서비스및 어플리케이션의 자동화된 빌드를 위하여 Jenkins를 사용
대표적인 CD 툴인 ArgoCD를 통해 자동화된 배포를 진행

argo

Jenkins

01 프로젝트 개요

프로젝트 환경 - Monitoring

EKS 클러스터 및 노드, 파드 등 여러 리소스들에 대한 모니터링 진행
Prometheus로 메트릭을 수집 후 Grafana로 시각화
AWS에서 사용 중인 리소스 모니터링을 위한 AWS CloudWatch 사용
Thanos, MinIO를 통한 Prometheus 고가용성 구현

02 프로젝트 팀 구성 및 역할

이름	역할	담당 업무
정한교	팀장	서비스 구현 관련 전반적인 개발, 팀원 조율
신현식	팀원	인프라 구축, CI/CD 파이프라인 설계 및 구축
최수환	팀원	인프라, CI/CD 파이프라인 설계 및 구축 , 배포용 git 관리
이승엽	팀원	인프라 구축, Monitoring 환경 구축, PPT 발표
배수빈	팀원	인프라 구축, Monitoring 환경 구축

02 프로젝트 팀 구성 및 역할

Phase 1 에서의 팀 구성

팀	이름	담당 업무
서비스팀	**정한교**	서비스 구현에 대한 기능 구성 기획, 테스트 코드 작성, 인프라팀에게 서비스 구현시 필요한 인프라 기술스택 전달
인프라팀	**신현식**	인프라 구축 및 RDS 인스턴스 구축 및 유지보수, 백엔드 테스트 코드 도커라이징 및 배포
	최수환	클러스터 배포 및 전반적인 인프라 설계 및 구축, 백엔드 테스트 코드 도커라이징 및 배포
	이승엽	인프라 구축, RDS 인스턴스 초기 설치 담당 프론트엔드 테스트 코드 도커라이징
	배수빈	인프라 설계 및 EFS 리소스 작성, 프론트엔드 테스트 코드 도커라이징

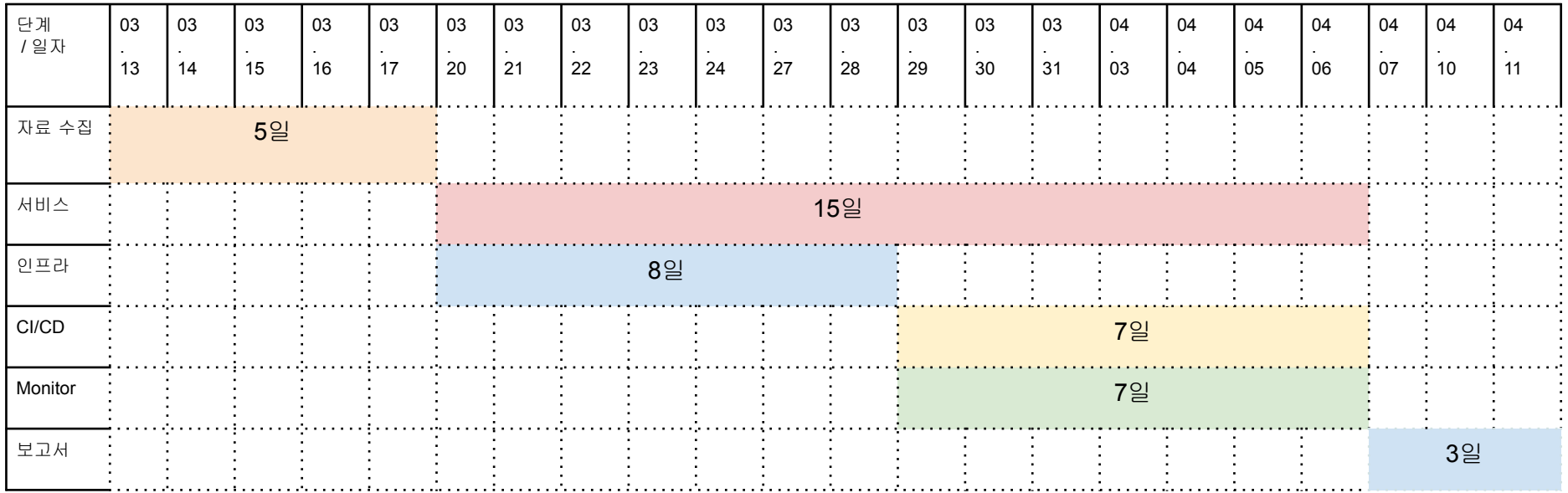
02 프로젝트 팀 구성 및 역할

Phase 2 에서의 팀 구성

팀	이름	담당 업무
서비스팀	**정한교**	서비스 프론트엔드 세부기능 구현,서비스 백엔드 세부기능 구현, 서비스파트 문서 작성, 서비스 파트 PPT 작성
CI/CD팀	**신현식**	인프라 설계 수정 및 변경사항 적용, 추가 인프라 구축 프론트엔드, 백엔드 CI/CD 파이프라인 구축 인프라, CI/CD 파트 기술문서, PPT 작성
	최수환	인프라 설계 수정 및 변경사항 적용, 추가 인프라 구축 프론트엔드 , 백엔드 CI/CD 파이프라인 구축 인프라, CI/CD 파트 기술문서, PPT 작성
모니터링팀	**이승엽**	Monitoring 환경 구축, Prometheus & Grafana 초기 설치, AWS CloudWatch 대시보드 작성, 기술문서 및 PPT 작성
	배수빈	Monitoring 환경 구축, Prometheus & Grafana 설치 및 구현, Thanos, MinIO를 통한 Prometheus HA 구현, 기술문서 및 PPT 작성

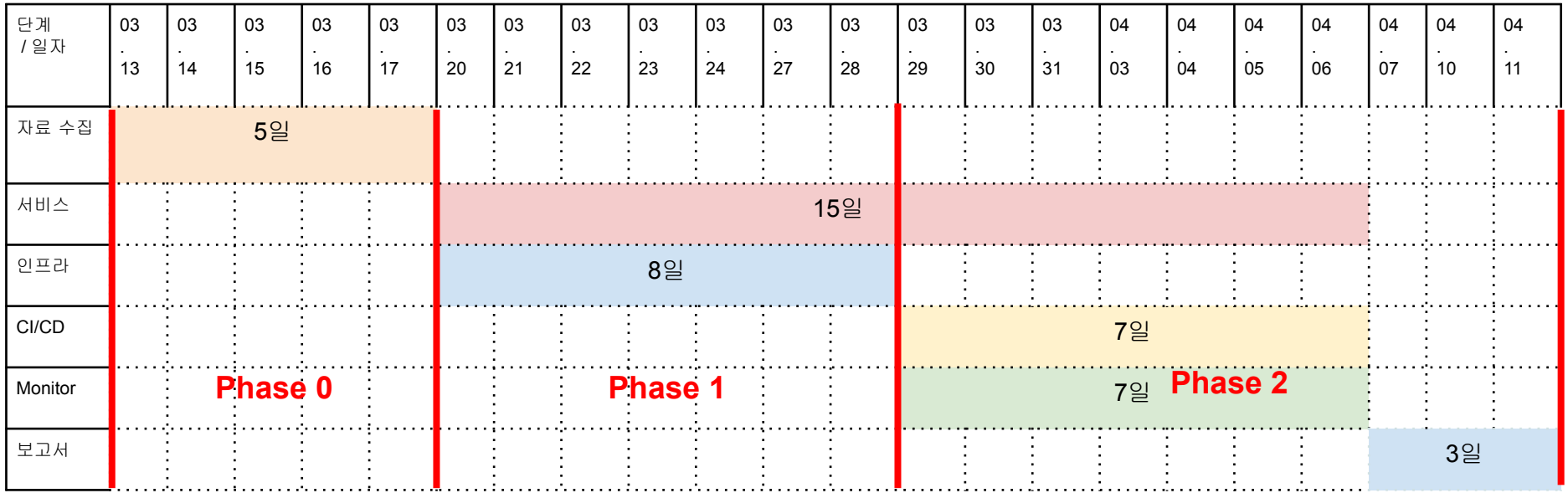
03 프로젝트 진행 프로세스

Phase 0~2 **Gantt chart** (03.20 ~ 04.11 **23일**)



03 프로젝트 진행 프로세스

Phase 0~2 **Gantt chart** (03.20 ~ 04.11 **23일**)



03 프로젝트 진행 프로세스

Phase 0 (03.13 ~ 03.17 **5일**)

자료 수집 및 기획 단계

- 자료 수집 및 인원 분배
- 각자 필요한 환경 및 기획 작성 (service, infra , cicd, monitor)
- service 기획에 맞게 infra 요구서 작성
- infra 요구서에 맞게 infra 기획서 작성
- 내용을 전부 모아서 프로젝트 착수 보고서 작성

단계 / 일자(2023년)	03.13	03.14	03.15	03.16	03.17
기획,자료수집	**5일**				

03 프로젝트 진행 프로세스

Phase 1 (03.20 ~ 03.28 **8일**)

기본적인 테스트 서비스 구현 및 기본적인 인프라 구현으로 테스트 서비스를 가동 단계
Phase 0 에서 진행한 기획에 맞게 기본적인 기능이 동작할 수 있는지에 대한 확인 작업

- infra 요구에 맞는 infrastructure 구성
- sevice test code 빌드 및 배포 (test frontend, test backent) 테스트
- infra 관련 network , authorization, storage 정리
- infra 관련 HA(High Availability) , security 구현

단계 / 일자	03.20	03.21	03.22	03.23	03.24	03.25	03.27	03.28
서비스구축	환경설정,기획(**2일**)		Backend test API, Frontend test Page(**6일**)					
인프라구축	AWS(EKS, RDS, EBS), service dockerizing, Permission Setting (**8일**)							

03 프로젝트 진행 프로세스

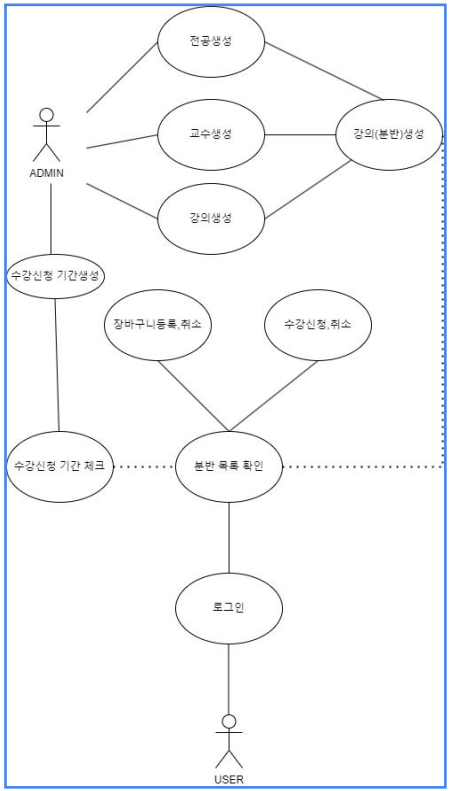
Phase 2 (03.29 ~ 04.11 **10일**)

기획에 따른 서비스 기능 개발 및 인프라 CI/CD , Monitoring 구현 및 보고서 작성 마무리 단계

- service 구현(backend api, frontend funtion)
- ci/cd pipeline 구현 (코드관련 jenkins, eks관련 argoCD)
- monitoring service 구현
- 작업 프로젝트 기술 문서, 발표PPT, 영상 작업

단계/일자	03.29	03.30	03.31	04.03	04.04	04.05	04.06	04.07	04.10	04.11
서비스	Back,Front 기능 구현 및 코드 정리 (**7일**)									
CI/CD	Argo CD 구현 및 Jenkins pipeline 구축(**7일**)									
Monitor	Prometheus, Grafana, Cloud watch 구축(**7일**)									
보고서								기술문서, PPT 제작(**3일**)		

04 프로젝트 결과 - 서비스



서비스 기획단계에서 구현한 Usecase Diagram

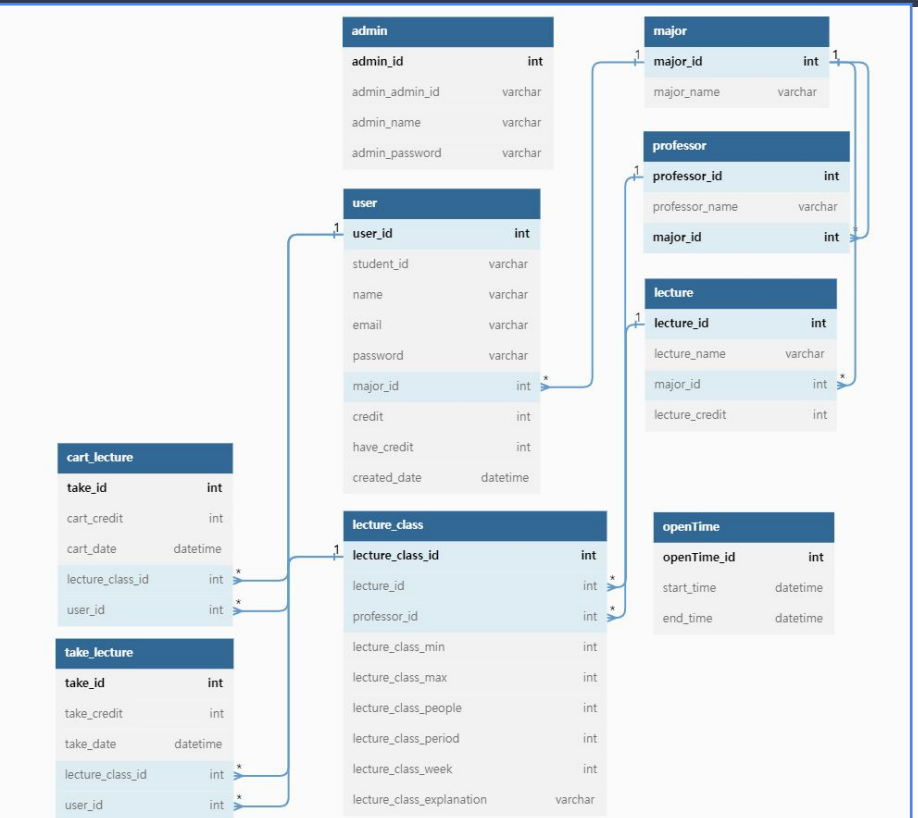
Admin (교직원)

- 로그인
- 전공생성
- 교수생성
- 강의생성 - > 강의(분반) 생성
- 수강신청 기간 생성

User (학생)

- 로그인
- 분반 목록 확인 -> 수강신청 기간 체크
- 장바구니 등록, 취소
- 수강등록, 취소

04 프로젝트 결과 - 서비스



DB (데이터베이스)

수강 신청 시스템과 관련해서 **관계형 데이터베이스** 가 데이터베이스를 표현하고 구현하는 데에 효율적

- **admin (교직원)**
- **user (학생)**
- **major (전공)**
- **professor (교수)**
- **lecture (강의)**
- **cart_lecture (장바구니)**
- **take_lecture (수강신청)**
- **lecture_class (강의분반)**
- **open_time (수강신청오픈시간)**

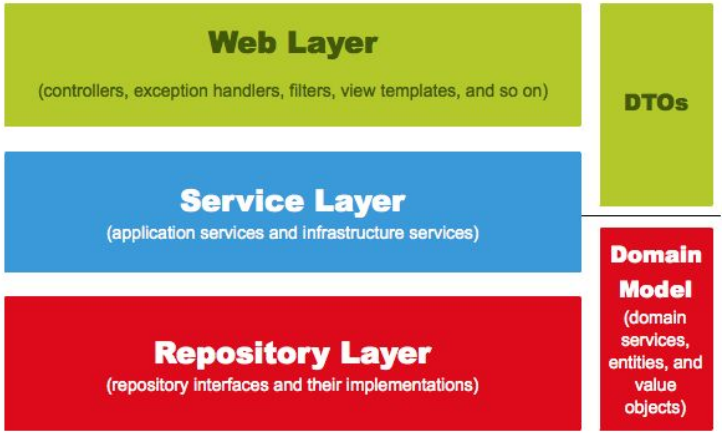
9개의 DB들이 서로서로 관계를 맺어 정보를 저장한다.

04 프로젝트 결과 - 서비스

Backend - Spring

앞서 Usecase와 DB의 정보를 CRUD하는 REST API를 스프링으로 구현

- **Web Layer (전반적인 외부 요청 응답)**
- **Service Layer (서비스 부분)**
- **Repository Layer (JPA 사용 저장소에 접근)**
- **DTOs (데이터 교환을 위한 객체)**
- **Domain Model (비즈니스 로직을 처리)**



스프링에서 권장하는 웹 5계층을 구분하여 개발을 하려고 노력하였다.
JWT를 사용한 로그인 보안 및 Spring security를 적용하여 비밀번호를 암호화 한다.

04 프로젝트 결과 - 서비스

Backend - Flask

MSA (MicroService Architecture)를 구현해보고자 하여 간단한 AI 서비스를 제공하는 백엔드 구현

Flask를 사용해 API 서버 구현

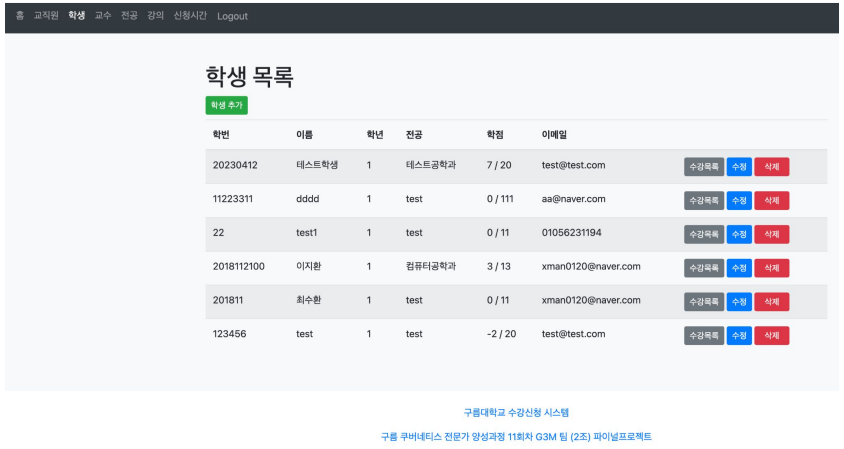
- **POST로 이미지를 받고**
- **pytorch와 이미 학습된 모델을 이용하여 이미지를 분석**
- **분석된이미지의 코드번호를 이미 저장된 json key로 찾고 value를 POST 응답**

직접 학습한 모델이 아닌 공개된 densenet121 모델을 사용하여 진행

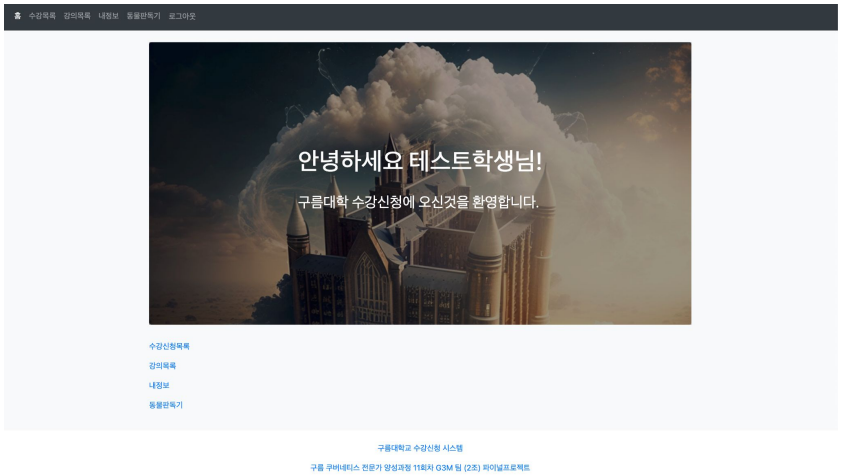
04 프로젝트 결과 - 서비스

Frontend - Vue

앞서 제작한 Backend의 API와 실제로 통신을하여 서비스의 기능을 사용자에게 보여주는 역할
Admin과 User가 구분되어 개발을 진행



Admin 학생 목록 탭



User 메인화면

04 프로젝트 결과 - 서비스

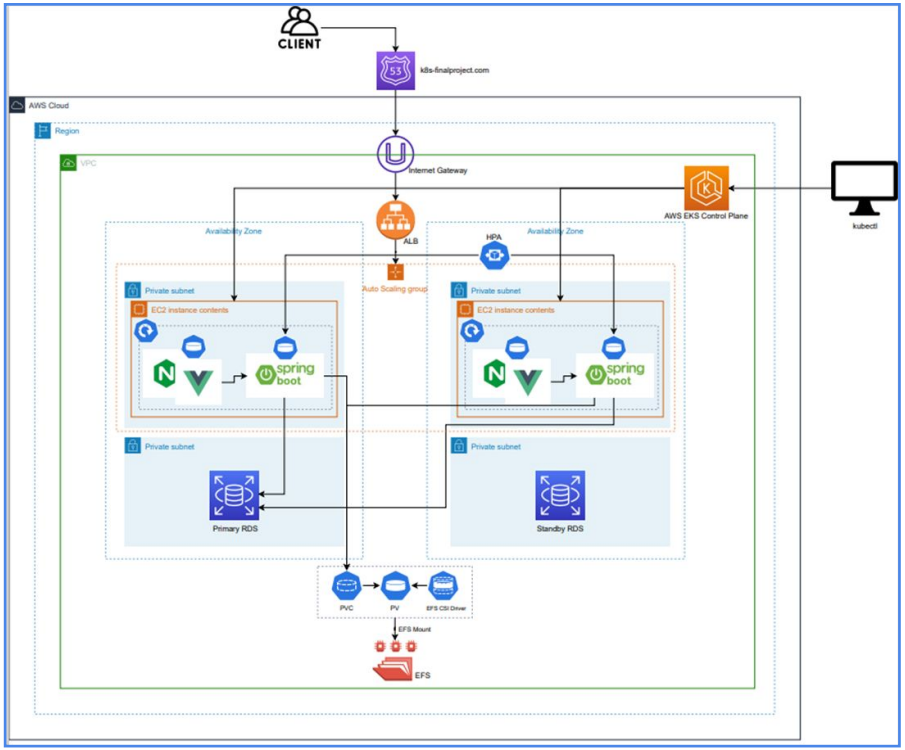
Frontend - Vue

앞서 제작한 Backend의 API와 실제로 통신을하여 서비스의 기능을 사용자에게 보여주는 역할
Vue를 이용해서 구현해 **실시간으로 바뀌는 서비스**를 보여주는데 탁월

User 수강신청 오픈 전

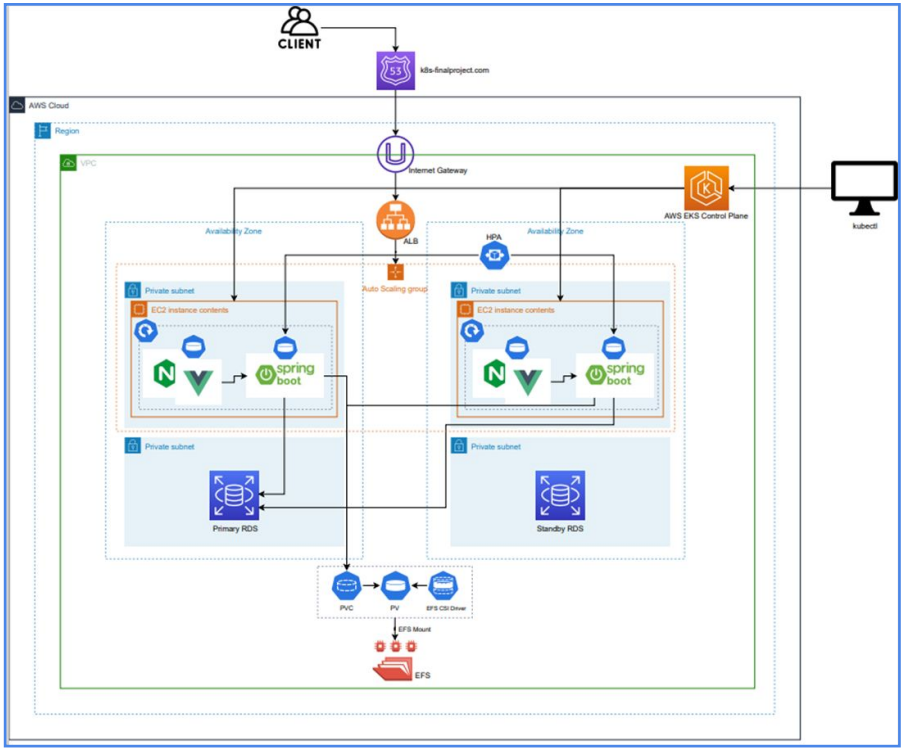
User 수강신청 오픈 후

04 프로젝트 결과 - 인프라 구축



- 각 노드에는 백엔드와 프론트엔드 파드가 Deployment에 의해 배포
- 고가용성을 위해 RDS 데이터베이스는 멀티 AZ 배포로 이중화 배치
- 파드가 HPA에 의해 늘어난 파드의 request 값을 해당 노드가 가진 한정된 자원으로는 늘어난 파드의 request 값을 보장해주지 못한다면 오토스케일링 그룹에 의해 EC2 인스턴스(=노드)가 증가
- 백엔드 서버 파드에 PVC를 부착하여 스토리지 클래스에 의해 동적으로 PV를 생성해 NFS 스토리지에 연결
  이때 EFS를 사용하여 EFS Provisionor에 PV를 연결시켜서 EFS에 마운트
- 인스턴스와 RDS는 보안을 위해 각각 프라이빗 서브넷에 배치하고 Ingress(ALB)와 NodePort Service(NLB)를 이용하여 외부의 클라이언트에 노출
- Route 53을 이용하여 외부 사용자가 도메인을 입력하여 서비스를 이용

04 프로젝트 결과 - 인프라 구축



Frontend - Backend - DB : 3-tier (3계층) 구조

- Frontend는 클라이언트가 보는 정적 페이지를 구축
- Backend는 서버의 역할로, 클라이언트가 정적 페이지와 상호 작용을 할 때 DB의 정보를 가지고 페이지가 계속 바뀌게끔 동적으로 구성
- DB는 RDS(관계형 데이터베이스)를 이용하여 정보를 체계적으로 관리

04 프로젝트 결과 - 인프라 구축

💡 EFS 스토리지 사용 이유

EFS는 자동 고성능 확장을 통해 여러 EC2 인스턴스에 대한 공유 파일 스토리지 옵션이 필요할 때마다 사용할 수 있다. 따라서 콘텐츠 관리 시스템을 위한 파일 스토리지로 적합하다. 우리는 고가용성을 위해 동일한 이미지를 가지고 있는 컨테이너를 여러 가용영역에 띄우도록 설계하였다.

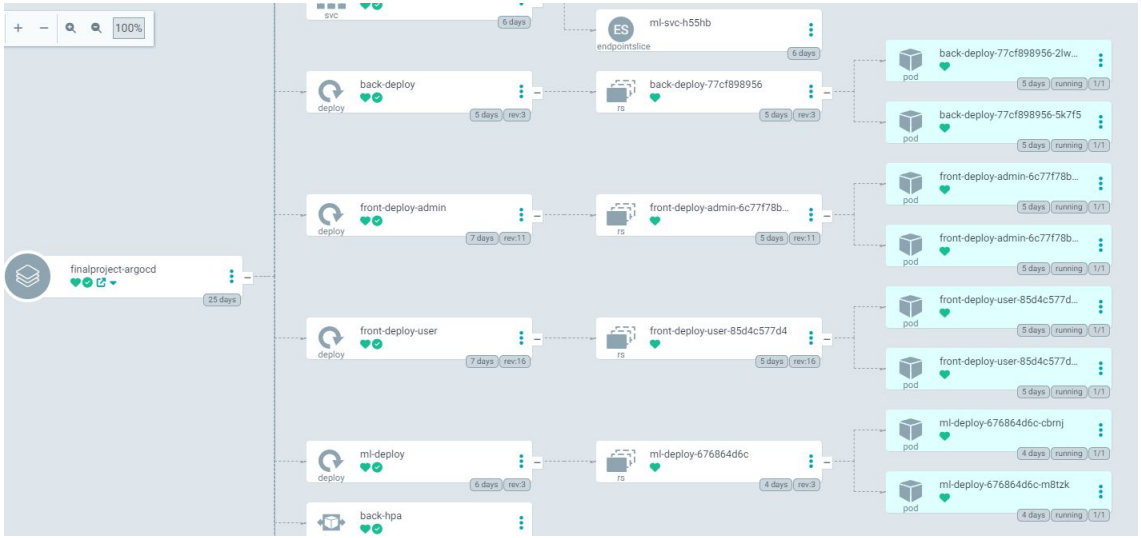
그렇기 때문에 같은 리전 내의 여러 가용 영역(AZ)에서 실행되는 Amazon EC2 인스턴스가 파일 시스템에 액세스할 수 있고, 공통 데이터 소스를 액세스 및 공유할 수 있기 때문에 EFS를 선택하였다.

💡 Backend에 HPA , Auto Scailing Group을 설정한 이유

기존의 EKS 클러스터는 Backend 서버를 위한 파드 두 개를 가동중이다. 하지만 수강신청 중 사용자의 수가 많아지고, 사용자의 요청이 증가함에 따라 Frontend에서는 Backend에 더 많은 요청을 보내게 될 것이다. Backend 파드의 CPU Utilization이 증가하게 될 것이고 그에 따라 고가용성을 위해 자동으로 Backend 파드의 개수를 늘려야 할 것이다. Backend 파드를 늘리지 못한다면 서비스의 다운타임이 발생할 것이고, 이를 방지하기 위해 HPA를 적용하였다.

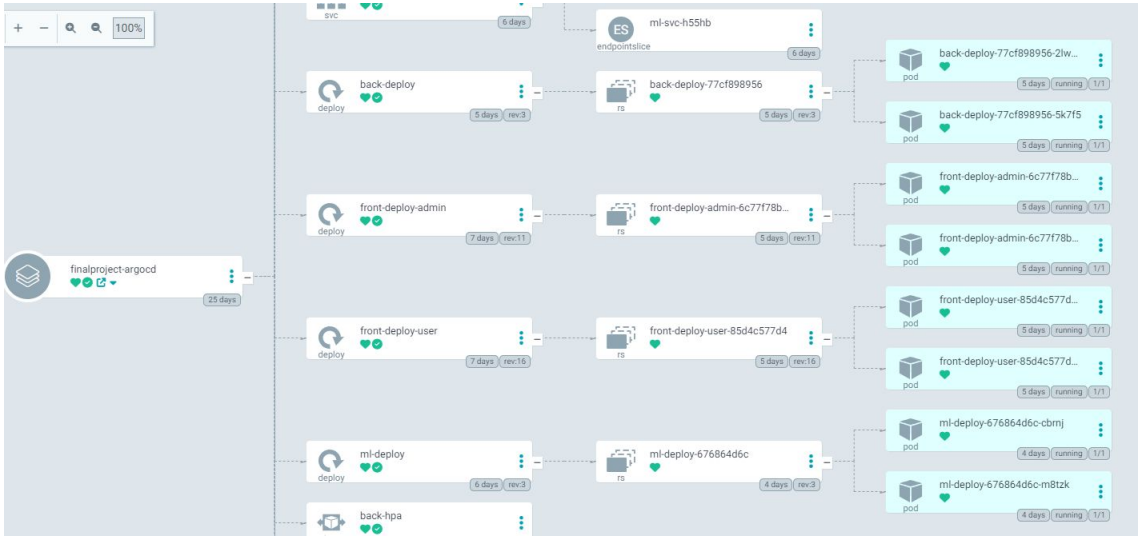
Backend 파드가 늘어남에 따라 EC2 인스턴스는 Backend 파드에 정해진 Request값을 보장해주어야 한다. 하지만 EC2 인스턴스가 늘어난 파드에 더이상 자신이 가진 자원으로 Request값을 보장해주지 못하게 된다면 이 또한 서비스의 다운타임을 발생시킬 것이다. 이를 방지하기 위해 EC2 인스턴스를 Auto Scailing Group으로 묶어주어 EC2 인스턴스의 자동확장을 설계하였다.

04 프로젝트 결과 - 인프라 구축



- frontend-user
  사용자가 접근하는 페이지를 위한 파드
- frontend-admin
  관리자가 접근하는 페이지를 위한 파드
- backend
  백엔드 요청을 처리하기 위한 파드
- ml
  머신러닝을 위한 파드

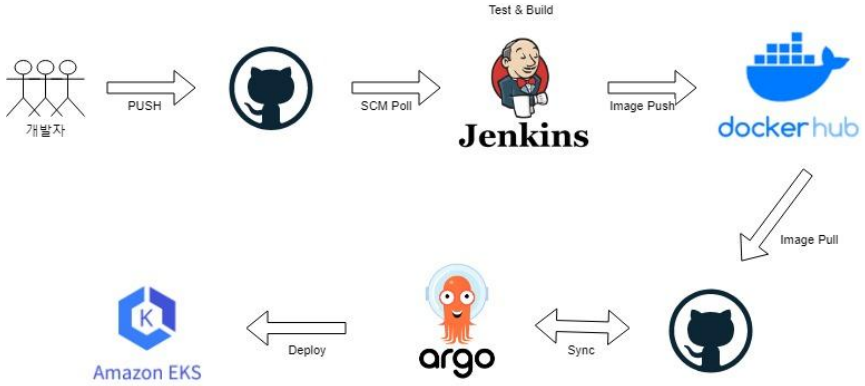
04 프로젝트 결과 - 인프라 구축



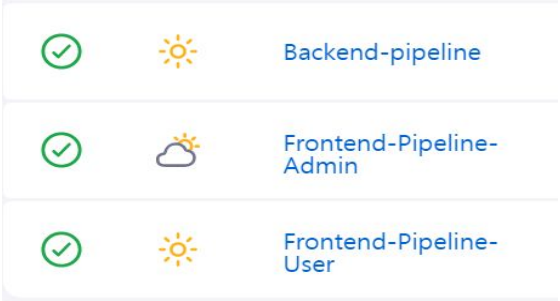
- 각 파드를 만들기 위한 deployment 생성
- Service 및 Ingress 설정
- backend 파드에 HPA 적용
- EFS 스토리지 생성
- Autoscailng 적용

04 프로젝트 결과 - CI/CD 파이프라인

CI/CD 파이프라인 아키텍처



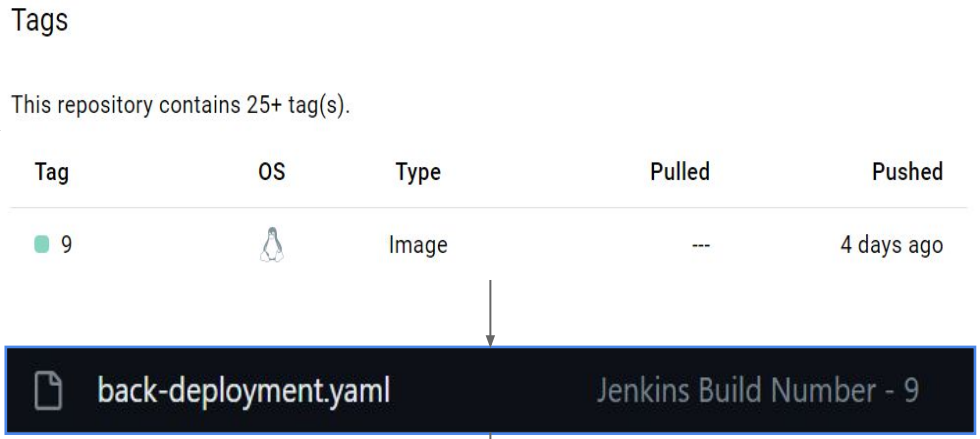
- Jenkins를 이용한 파이프라인 구축
- Jenkins와 연동되어 빌드될 개발자의 코드가 존재하는 git과 ArgoCD와 연동되어 배포될 git을 구분
- 이미지 저장소로 Docker Hub사용
- Backend , Frontend-User , Frontend-Admin 총 3개의 파이프라인이 존재
- 각 파이프라인의 아키텍처는 동일

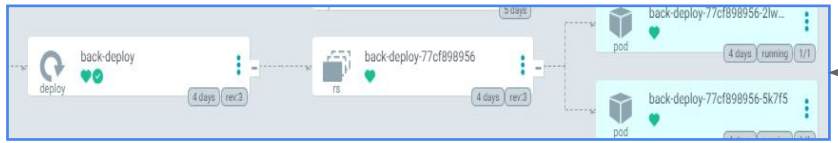


- 각 파이프라인과 연결된 git에는 Jenkinsfile과 Dockerfile이 존재

04 프로젝트 결과 - CI/CD 파이프라인

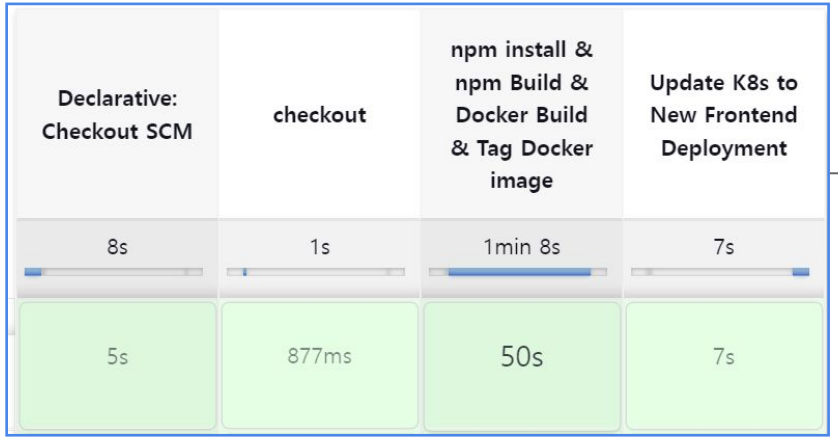
Backend 파이프라인

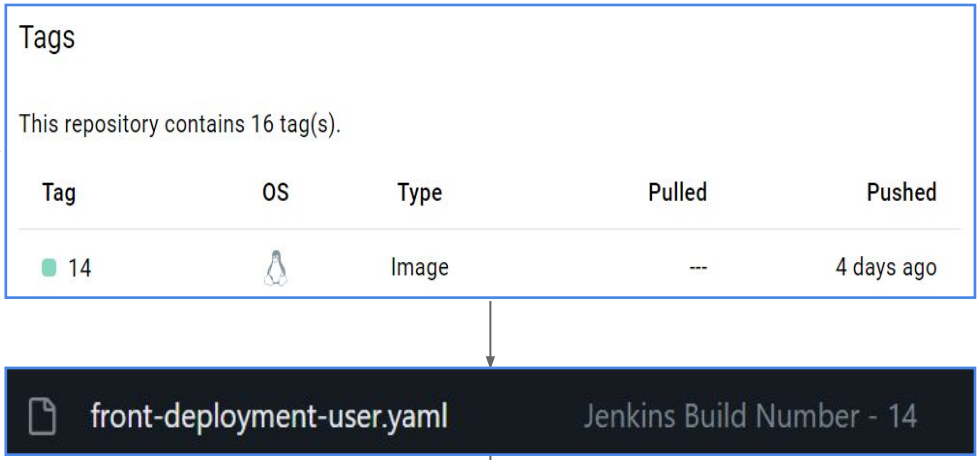




04 프로젝트 결과 - CI/CD 파이프라인

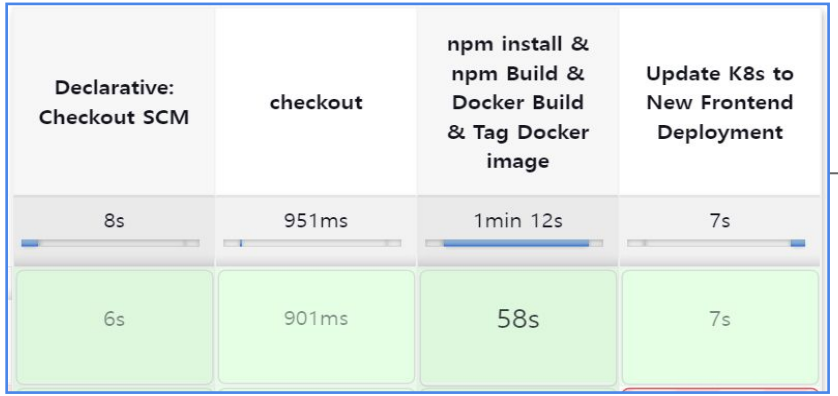
Frontend - User 파이프라인

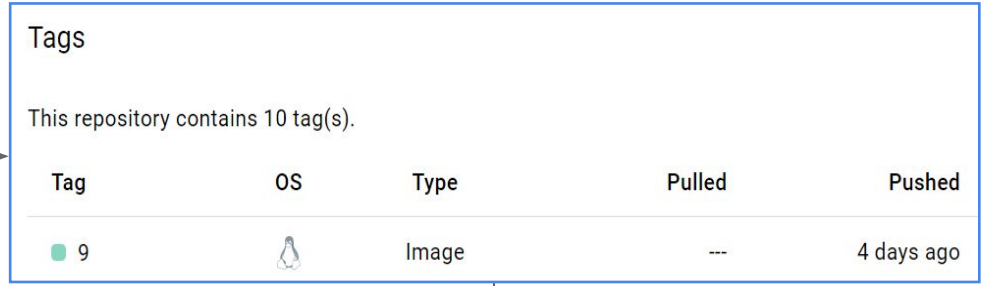


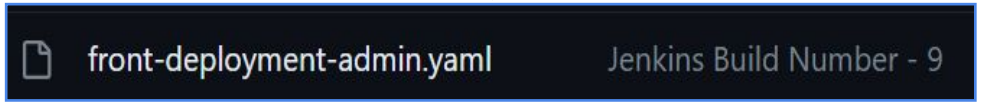


04 프로젝트 결과 - CI/CD 파이프라인

Frontend - Admin 파이프라인

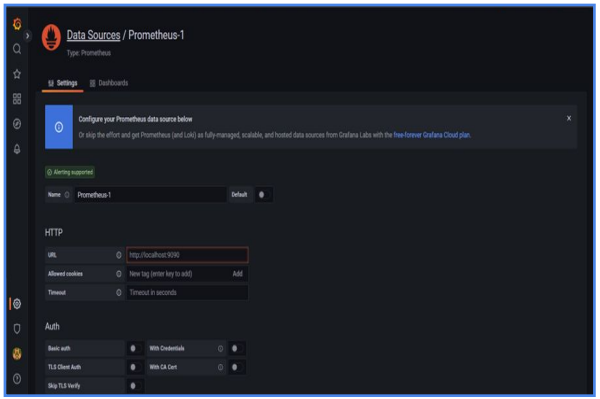


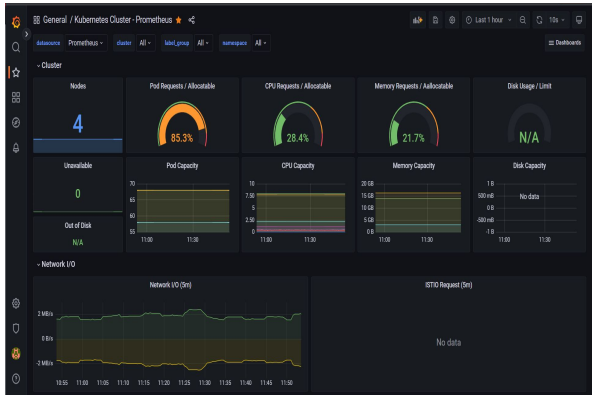




04 프로젝트 결과 - Monitoring

Prometheus & Grafana





- kube-prometheus-stack을 이용한 클러스터 **내부 배치**
- LoadBalancer로 지정한 외부IP 주소를 통한 Grafana 접속
- Prometheus Data source 추가 후 Grafana 대시보드 구성

04 프로젝트 결과 - Monitoring

Prometheus HA

단순 프로메테우스 서버 복제 시 발생 문제

- 메트릭 데이터 중복문제
- Stateful 구성으로 인한 서버간 데이터 공유 문제

→ **“Thanos와 MinIO를 통한 모니터링의 고가용성 확보”**

04 프로젝트 결과 - Monitoring

Prometheus HA

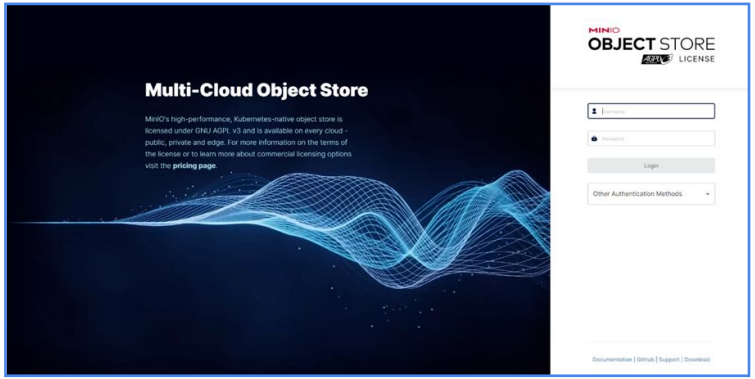
- Prometheus 서버 데이터 중앙 집중화 저장
- 고가용성 분산 시스템 환경에서 지연시간 최적화

- Prometheus 데이터를 저장할 Object Storage
- 데이터 처리 및 보존성, 확장성, 비용 측면의 이점

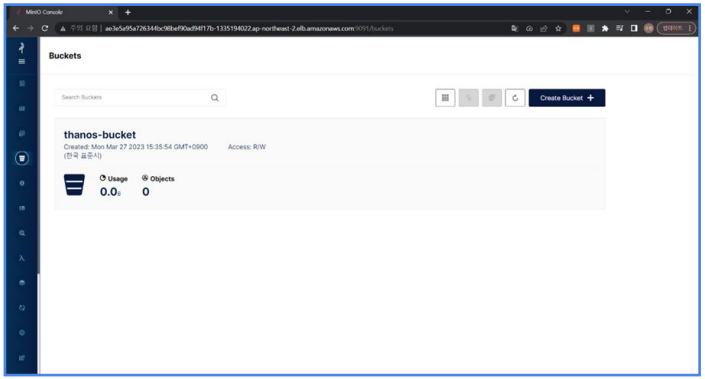
04 프로젝트 결과 - Monitoring

Prometheus HA - MinIO 구성

MinIO의 볼륨 : 프로젝트에서 사용하는 EFS
PVC 생성 : Monitoring 네임스페이스에서 EFS 사용
서비스 생성 : MinIO 웹 페이지 연결



http://<minio-svc External-IP>:9091 MinIO 사이트 접속



Thanos에서 사용될 thanos-bucket 생성

04 프로젝트 결과 - Monitoring

Prometheus HA - Thanos 구성을 위한 기본구성

Values.yaml 커스텀

- **Secret 생성**
  : Thanos 와 MinIO 연결

- **Thanos sidecar 설정**
  : Prometheus 서버에서 수집한 데이터 Thanos Query, Thanos Store 전송

  values.yaml 파일 추가 설정으로 구현

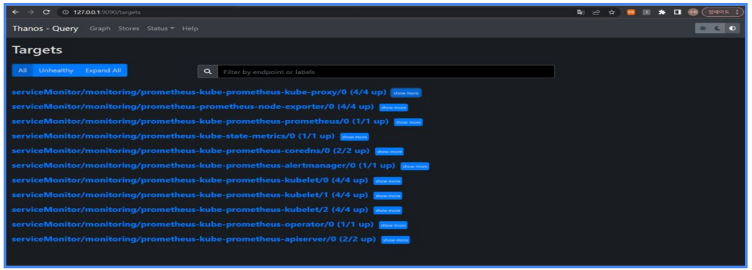
```
my-prometheus-kube-prometh-thanos-discovery   LoadBalancer   10.100.81.36     a5b2c982c65ca4f69bfc97df9937640a-661505608.ap-northeast-2.elb.amazonaws.com
  10901:31933/TCP,10902:30088/TCP   25h
```

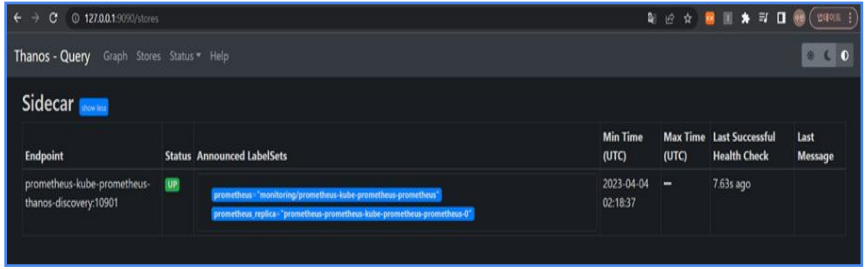
04 프로젝트 결과 - Monitoring

Prometheus HA - Thanos 설치

thanos.yaml : Thanos 설치 시 필요한 내용을 추가한 사용자 정의 Thanos Helm Chart 구성 파일 작성

Thanos 설치 : bitnami/thanos를 사용한 helm chart로 설치

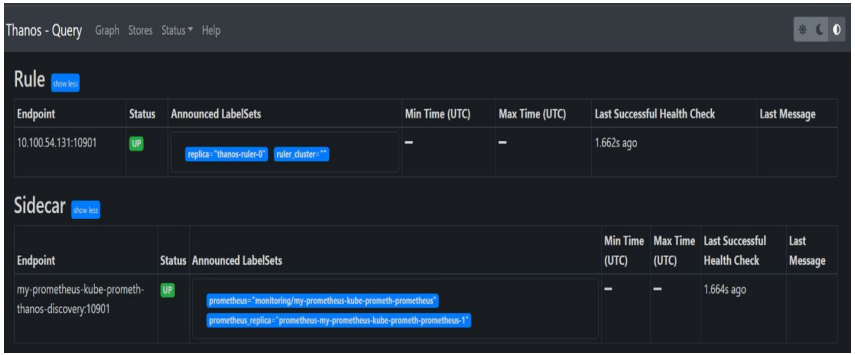


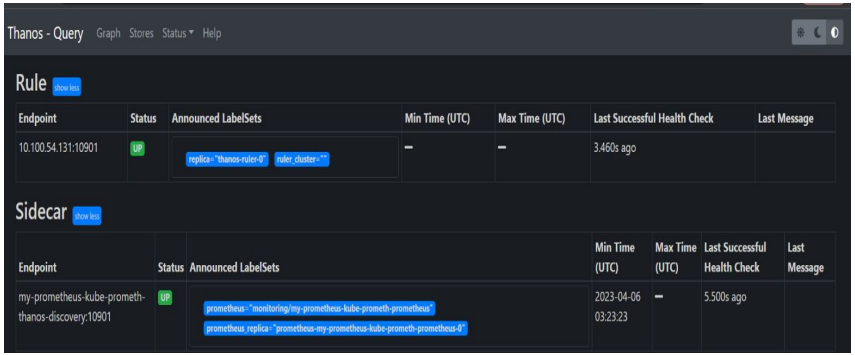


- Thanos – Query 웹 사이트의 targets에서 Prometheus 메트릭 성공적으로 받아오고 있는 모습 확인
- Thanos-Sidecar 정상동작 확인

04 프로젝트 결과 - Monitoring

Prometheus HA - Prometheus 서비스 이중화





- **Prometheus 서비스 추가**
- **Headless 서비스 생성** : 추가 생성한 Prometheus 서비스 Pod를 Thanos와 연결
- **thanos.yaml 파일 업데이트** : querier.stores파트 headless 서비스 추가

→ **prometheus-my-prometheus-kube-prometh-prometheus-0, prometheus-my-prometheus-kube-prometh-prometheus-1 번갈아가며 사용 확인**

04 프로젝트 결과 - Monitoring

- AWS 내 리소스들의 현황을 모니터링하기 위해 AWS CloudWatch 웹 콘솔에 접속한다.
- AWS 내 리소스들을 모니터링하기 위해서는 AWS CloudWatch의 대시보드가 필요하다.
- 우측 상단의 '대시보드 생성' 버튼을 클릭하여 새 대시보드를 생성한다.

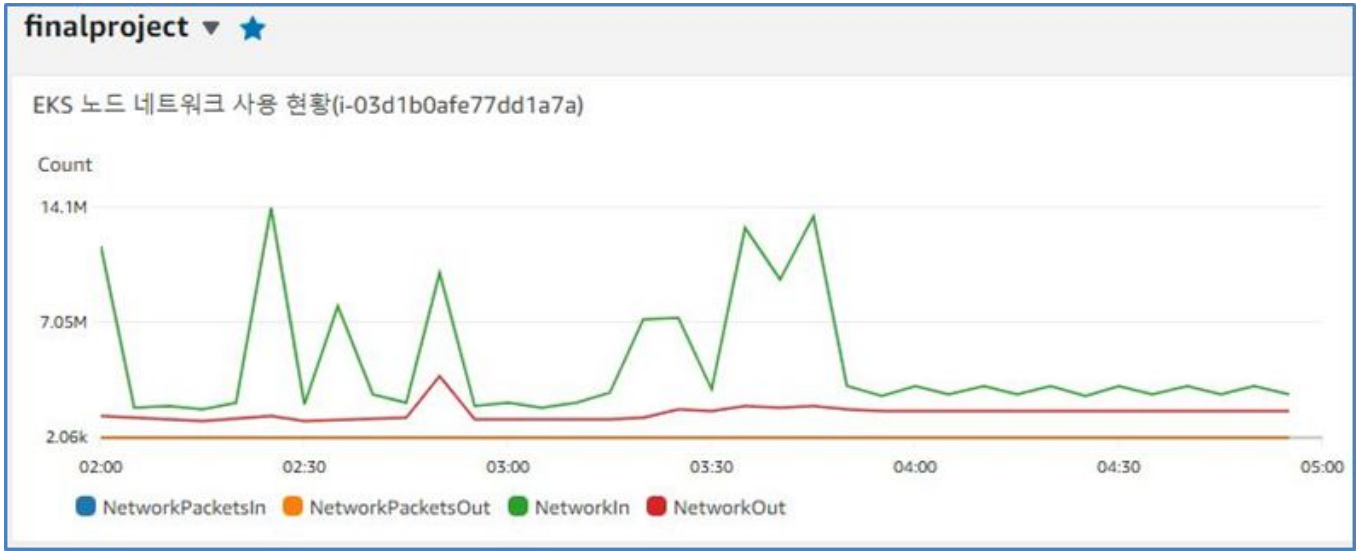
04 프로젝트 결과 - Monitoring

- ‘대시보드 생성’ 버튼을 누르면, 대시보드의 이름을 정하는 창이 출력된다.
- 대시보드에 원하는 이름을 적고, ‘대시보드 생성’ 버튼을 클릭한다.
- 새로 생성된 대시보드는 아무 요소도 포함되지 않은 순정 상태의 대시보드로 생성된다.

04 프로젝트 결과 - Monitoring

- 모니터링하고자 하는 지표에 맞는 위젯을 취사선택하여 배치하도록 한다.

04 프로젝트 결과 - Monitoring



- 위 사진은 '행' 위젯으로 표시되는 EKS 클러스터 노드들 중 한 노드의 네트워크 현황
- '행' 위젯을 통해 네트워크 변동 상황을 실시간으로 모니터링 가능

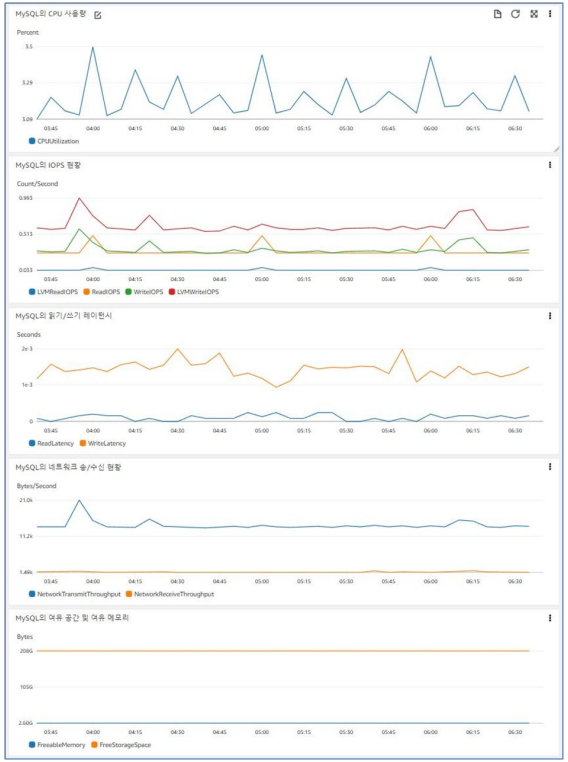
04 프로젝트 결과 - Monitoring

- 위 사진은 '게이지' 위젯으로 표시되는 EKS 클러스터 노드들의 EBS 볼륨 사용 현황
- 점유율, 사용량과 같은 값들을 사용량/여유 공간의 형식으로 시각화해주는 위젯
- 별도로 게이지의 최대값과 최소값을 옵션 탭에서 지정해 주어야 사용 가능

04 프로젝트 결과 - Monitoring

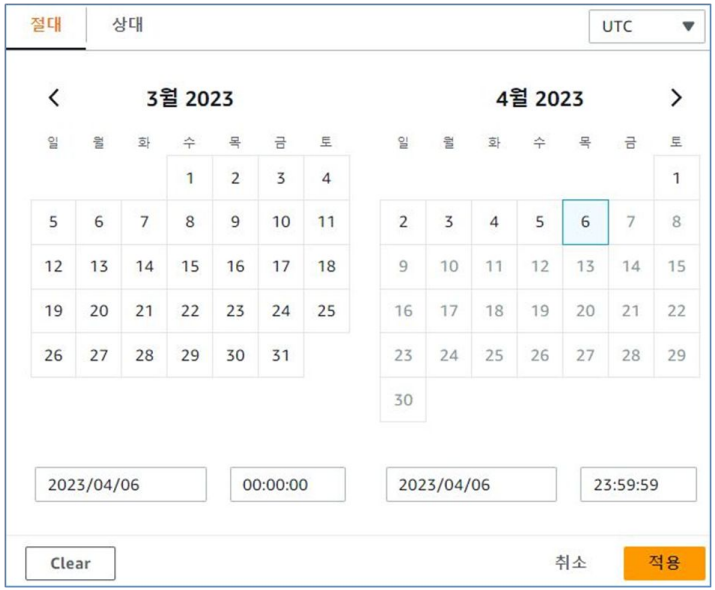
- 위 사진은 '번호' 위젯을 사용하여 표시한 EFS의 사용량
- 값에 사용되는 단위 역시 개별적으로 지정 가능

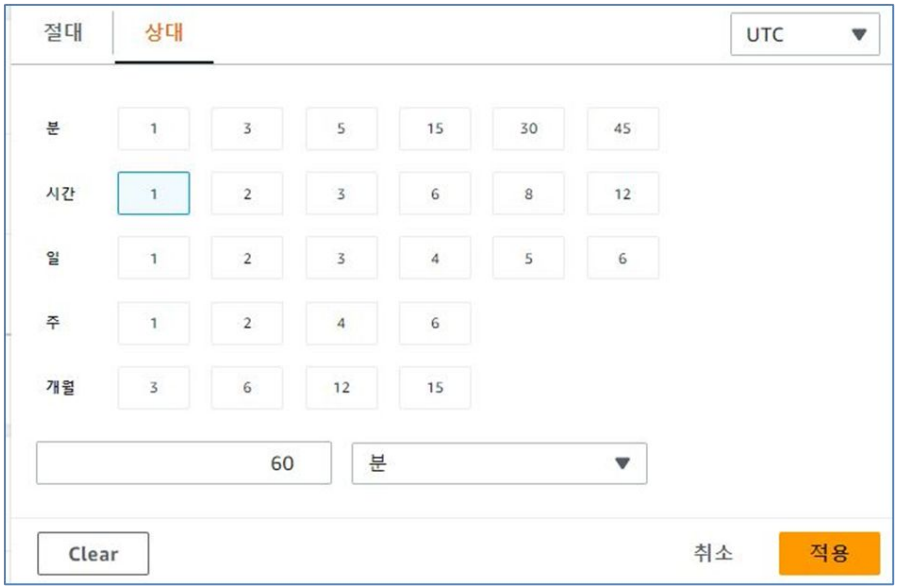
04 프로젝트 결과 - Monitoring



- 좌측 사진은 '행' 위젯을 사용하여 표시되고 있는 RDS 인스턴스 현황
- 각각 MySQL 내부 요소들의 변동 사항을 실시간으로 표시 중
- 현재 CPU 사용량, IOPS 현황, 읽기/쓰기 레이턴시, 네트워크 송/수신 현황, 여유 공간 및 여유 메모리 표시 중

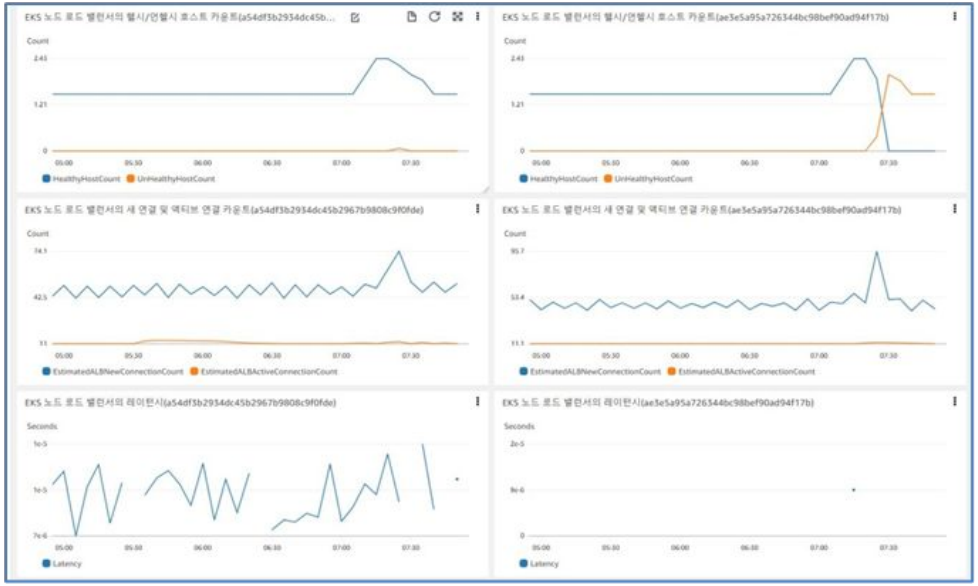
04 프로젝트 결과 - Monitoring





- 원하는 시간대를 지정하여 표시시키는 방법이 존재
- 시간 탭에서 '사용자 지정' 버튼을 누른 뒤 '절대' 탭을 눌러 원하는 시작 시간과 끝 시간을 지정 가능
- 현재 시간을 기준으로 얼마나 전 시간대의 값을 조회할 것인지 '상대' 탭을 통해 개별 선택 가능

04 프로젝트 결과 - Monitoring



- 위 사진은 EKS 노드에서 사용 중인 로드 밸런서의 네트워크 변동 사항을 표시 중인 '행' 위젯
- 각각 로드 밸런서의 헬시/언헬시 호스트 카운트, 새 연결 및 액티브 연결 카운트, 레이턴시를 표시 중
- 동일한 방법으로 Grafana의 로드 밸런서 또한 표시 중

05 자체 평가 및 보완

1. *전체적인 서비스가 무거움*
2. *서비스 **Page** 디자인 개선*
3. ***Machine Learning** 서버 파이프라인 구축*
4. ***AWS CloudWatch** 지표 추적 및 갱신 자동화 고안*
5. ***Thanos**를 이용한 완벽한 **Prometheus HA** 구현*

감사합니다

구름 쿠버네티스 전문가 과정 11회차

TEAM G3M