구름대학 수강신청 프로젝트

팀명 : **G3M**

팀장 : 정한교

팀원 : 이승엽
최수환
배수빈
신현식

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

목 차

1. 프로젝트 개요 **10**
1.1. 프로젝트 목표 10
1.1.1. 주제 개요 및 목표 10
1.1.2. 주제 선정 배경 11
1.2. 팀원 및 역할 12
1.3. 프로젝트 일정 및 진행 프로세스 13
1.3.1. Phase 0 (03.13 ~ 03.17 5일) 13
1.3.2. Phase 1 (03.20 ~ 03.28 8일) 13
1.3.3. Phase 2 (03.29 ~ 04.11 10일) 14
1.3.4. Phase 0~2 Gantt chart 14
1.4. 프로젝트 환경 15
1.4.1. Service ENV 15
1.4.1.1. Database 15
1.4.1.2. Backend 15
1.4.1.3. Frontend 16
1.4.2. Infrastructure ENV 17
1.4.3. CI/CD ENV 18
1.4.4. Monitoring ENV 19
1.4.5. 프로젝트 관리 방안 20
1.5. 프로젝트 기대효과 21
1.5.1. 현황 (AS-IS) 21
1.5.2. 개선 및 기대효과 (TO-BE) 21
2. 프로젝트 내용 **22**
2.1. Service 22
2.1.1. 서비스 기획 및 설계 22
2.1.1.1. 배경 및 목표 22
2.1.1.2. 서비스 진행 프로세스 22
2.1.1.3. 서비스 기획(기능정리) 22
2.1.1.4. 서비스 기획(Use case) 23
2.1.2. DB 24
2.1.2.1. 데이터베이스 환경 24
2.1.2.2. 데이터베이스 구성 25
2.1.2.2.1. Admin Table 26

2.1.2.2.2. User Table 27
2.1.2.2.3. Major Table 28
2.1.2.2.4. Professor Table 28
2.1.2.2.5. Lecture Table 29
2.1.2.2.6. Lecture_Class Table 30
2.1.2.2.7. Take_Lecture Table 31
2.1.2.2.8. Cart_Lecture Table 32
2.1.2.2.9. Open_Time Table 33
2.1.2.3. 데이터베이스 코드 34
2.1.3. Backend 37
2.1.3.1. Backend 환경 37
2.1.3.2. Backend 구성 37
2.1.3.3. Backend (Spring) 38
2.1.3.3.1. Web Layer 38
2.1.3.3.2. Service Layer 39
2.1.3.3.3. Repository Layer 40
2.1.3.3.4. DTOs 41
2.1.3.3.5. Domain model 42
2.1.3.3. Backend (Flask) 45
2.1.3.4. Backend API 모음 47
2.1.4. Frontend 49
2.1.4.1. Frontend 환경 49
2.1.4.2. Frontend 구성 50
2.1.4.3. Frontend (admin) 51
2.1.4.4. Frontend (user) 56
2.2. infrastructure 62
2.2.1. infrastructure 기획 62
2.2.1.1. infrastructure architecture 설계 62
2.2.2. infrastructure 구축 64
2.2.2.1. EKS cluster 64
2.2.2.2. Service deploy 67
2.2.2.2.1. Frontend (Vue-user) deploy 68
2.2.2.2.2. Frontend (Vue-Admin) deploy 70
2.2.2.2.3. Frontend Ingress 72
2.2.2.2.4. Backend (Java Spring boot) deploy 73
2.2.2.2.5. Backend (Python Flask) deploy 76
2.2.2.3. EFS Storage 78
2.2.2.4. HPA 79

2.2.2.5. ArgoCD....81
2.2.2.6. EKS-RDS 연동 및 생성....82
2.2.3. CI/CD....94
2.2.3.1. pipeline architecture 설계....94
2.2.3.2. Backend Pipeline 구축....96
2.2.3.2.1. Backend 도커라이징....96
2.2.3.2.1.1. Backend - RDS연동....97
2.2.3.2.1.2. Dockerfile....97
2.2.3.2.2. Jenkins를 이용한 Pipeline....98
2.2.3.2.2.1. Jenkinsfile....98
2.2.3.2.2.2. Pipeline 결과 테스트....101
2.2.3.3. Frontend - User Pipeline 구축....102
2.2.3.3.1. nginx에서 Frontend - Backend 연동....102
2.2.3.3.2. Frontend 설정파일에서 Frontend - Backend 연동....103
2.2.3.3.3. Frontend - User 도커라이징....104
2.2.3.3.4. Jenkins를 이용한 Pipeline....106
2.2.3.3.4.1. Pipeline 결과 테스트....109
2.2.3.4. Frontend - Admin Pipeline 구축....110
2.2.3.4.1. nginx에서 Frontend - Backend 연결....110
2.2.3.4.2. Frontend 설정파일에서 Frontend - Backend 연결....111
2.2.3.4.3. Frontend - Admin 도커라이징....111
2.2.3.4.4. Jenkins를 이용한 Pipeline....112
2.2.3.4.4.1. Pipeline 결과 테스트....114
2.2.4. Monitoring....116
2.2.4.1. Prometheus & Grafana....116
2.2.4.1.1. Prometheus & Grafana 배포....116
2.2.4.1.2. Grafana 접속 및 Dashboard 구성....117
2.2.4.2. Prometheus HA....119
2.2.4.2.1. Object Storage 구성....119
2.2.4.2.2. Secret....124
2.2.4.2.3. Thanos sidecar 설정....125
2.2.4.2.4. Thanos 설치....125
2.2.4.2.5. Prometheus HA 구성....127
2.2.4.3. Amazon CloudWatch....130
2.2.4.3.1. Amazon CloudWatch 대시보드 구성....130
3. 결론**....139**
4. 추후과제 및 후기**....141**
4.1. 추후 과제....141

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

4.2. 후기 142
5. 참고 자료 **..... 144**

표 목차

표 1 팀원 구성과 역할 및 담당 업무
표 2 Phase 0
표 3 Phase 1
표 4 Phase 2
표 5 Gantt chart
표 6 Database env
표 7 Backend env
표 8 Frontend env
표 9 Infrastructure ENV
표 10 Monitoring ENV
표 11 프로젝트에서 사용한 DB system
표 12 Admin Table
표 13 User Table
표 14 Major Table
표 15 Professor Table
표 16 Lecture Table
표 17 Lecture_Class Table
표 18 Take_Lecture Table
표 19 Cart_Lecture Table
표 20 Open_Time Table
표 21 Backend 환경
표 22 Web layer
표 23 Web layer
표 24 repository class
표 25 Frontend 도구

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

그림 목차

그림 1 service usecase
그림 2 DB diagram
그림 3 Admin Table
그림 4 User Table
그림 5 Major Table
그림 6 Professor Table
그림 7 Lecture Table
그림 8 Lecture_Class Table
그림 9 Take_Lecture Table
그림 10 Cart_Lecture Table
그림 11 Open_Time Table
그림 12 로그인
그림 13 회원가입
그림 14 메인페이지
그림 15 교직원
그림 16 학생
그림 17 교수
그림 18 전공
그림 19 수업
그림 20 수강신청개시시간
그림 21 User 로그인
그림 22 User 메인페이지
그림 23 User 마이페이지
그림 24 User 장바구니 및 수강목록
그림 25 User 강의 목록
그림 26 User 동물판독기
그림 27 infrastructure architecture
그림 28 cluster.yaml (1)
그림 29 cluster.yaml (2)
그림 30 cluster.yaml (3)
그림 31 front-deployment-user
그림 32 front-svc-user
그림 33 front-deployment-admin
그림 34 front-admin-svc
그림 35 front-ing
그림 36 back-deployment
그림 37 back-svc

그림 38 back-ing
그림 39 ml-deployment
그림 40 ml-svc
그림 41 ml-ing
그림 42 efs-storage-class
그림 43 efs-pvc
그림 44 back-hpa2.2.2.5 Cluster Autoscaler
그림 45 Cluster Autoscaler
그림 46 ArgoCD command
그림 47 ArgoCD Dashborad
그림 48 EKS 정보
그림 49 보안그룹 생성
그림 50 RDS용 서브넷 생성 (1)
그림 51 RDS용 서브넷 생성 (2)
그림 52 RDS 서브넷 그룹 생성
그림 53 파라미터 그룹 생성 (1)
그림 54 파라미터 그룹 생성(2)
그림 55 RDS 생성
그림 56 라우팅 테이블 작성
그림 57 외부에서 MySQL 접속 (1)
그림 58 외부에서 MySQL 접속 (2)
그림 59 Pipeline Architecture
그림 60 pipeline
그림 61 Whitelabel Error Page
그림 62 Pipeline 결과 테스트
그림 63 Docker Hub Image
그림 64 학생 로그인
그림 65 Pipeline 결과 테스트 1
그림 66 Frontend - User Pipeline
그림 67 Admin 로그인
그림 68 Pipeline 결과 테스트 2
그림 69 서비스 확인
그림 70 Grafana webserver
그림 71 Prometheus Data source 추가
그림 72 Dashboard
그림 73 MinIO pod 생성
그림 74 MinIO 웹 페이지 접속
그림 75 MinIO 버킷 (thanos-bucket) 생성
그림 76 thanos-obstore-secret-v1 시크릿 확인
그림 77 Thanos sidecar 생성 (my-prometheus-kube-thanos-discovery)
그림 78 thanos-query의 targets 확인
그림 79 thanos-sidecar 확인

그림 80 2개의 prometheus 서비스 생성
그림 81 Prometheus HA 설정 후 Sidecar 모습(1)
그림 82 Prometheus HA 설정 후 Sidecar 모습(2)
그림 83 대시보드 생성
그림 84 대시보드 이름 지정
그림 85 대시보드에 포함할 위젯 생성
그림 86 '행' 위젯으로 볼 수 있는 EKS 노드의 네트워크 사용 현황
그림 87 EKS 클러스터 노드들의 개별 EBS 볼륨 사용 현황
그림 88 '게이지' 위젯 옵션 중 '게이지 범위'
그림 89 '번호' 위젯으로 조회되는 Amazon Elastic File System(EFS)의 사용량
그림 90 '행' 위젯으로 표시되고 있는 RDS 인스턴스 내 MySQL의 상황
그림 91 표시 시간 단위
그림 92 원하는 시간대를 직접 지정하여 값을 조회하는 방법
그림 93 현재 시간을 기준으로 조회할 값들을 정하는 방법
그림 94 EKS 노드 로드 밸런서의 지표 현황
그림 95 그라파나 로드 밸런서의 지표 현황

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

1. 프로젝트 개요

1.1. 프로젝트 목표

1.1.1. 주제 개요 및 목표

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.

수강신청 시스템의 존재 의의는 수강신청 기간에 있다고 볼 수 있는데 그 수강신청 기간에 동시에 수강신청을 하거나, 수업 정보를 조회하는 등의 작업으로 인해 서버 부하증가 및 서버 지연으로 시스템 자체가 다운되거나 느려지는 등의 문제가 발생한다.

그렇기에 G3M 팀은
안정적인 인프라를 구축하고, 대규모 트래픽을 처리할 수 있는 서비스를 구현한다.

대학교 수강신청 서비스 프로젝트에서는 안정적인 인프라를 구축하고, 대규모 트래픽을 처리할 수 있는 시스템을 구현하는 것을 목표로 하고 이를 위해 클라우드 컴퓨팅, 분산 시스템, 데이터베이스 성능 최적화 등의 기술을 활용하여 시스템의 안정성과 확장성을 높인다.

또한, 보안 측면에서도 사용자 정보를 안전하게 보호하고, 시스템이 해킹이나 악성 공격 등의 공격으로부터 안전하게 보호될 수 있도록 인프라를 설계한다.

1.1.2. 주제 선정 배경

수강신청 서비스란

기존의 대학교 수강신청 시스템에서는 학생들이 동시에 수강신청을 하거나, 수업 정보를 조회하는 등의 작업으로 인해 서버 부하가 증가하게 되면 시스템이 다운되거나 느려지는 등의 문제가 발생할 수 있고. 이로 인해 학생들은 수강신청에 실패하거나, 수업 정보를 제대로 파악하지 못하게 될 가능성이 있으며, 수강 신청 시간에 동시 접속자들이 많다 보니 서버가 지연되어 신청을 했는데도 신청 처리가 바로 되지 않거나 초기화가 되는 일이 발생한다는 것이다. 학생들이 수강 신청을 할 때 긴장을 하게 되는 가장 큰 이유이기도 하다. 수강 신청이 학생들의 한 학기 동안의 학습 환경에 막대한 영향을 주는 만큼 대학은 모든 학생이 강의를 안정적으로 신청할 수 있도록 사이트의 서버를 관리할 책임도 있다. 대학은 기존 학생 수 이상의 접속자를 수용할 수 있도록 원활한 인프라를 구축해야 한다.

주제 선정 이유

주제를 선정한 이유로는 G3M팀은 인프라 구축을 전문으로 하는 팀이다.
대학교 수강신청 서비스란 인프라에 관하여 많은 문제가 발생하는 서비스 중에 하나라고 생각했다. 회의 결과 G3M 팀은 수강신청 서비스를 안정적이게 서비스를 제공할 수 있으며, 수강신청과 관련된 많은 문제를 해결할 수 있다고 생각이 되어서 이 주제를 선정하게 되었다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

1.2. 팀원 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

Phase 1

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

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

Phase 2

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

표 1 팀원 구성과 역할 및 담당 업무

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

1.3. 프로젝트 일정 및 진행 프로세스

1.3.1. Phase 0 (03.13 ~ 03.17 **5**일)

자료 수집 및 기획 단계

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

표 2 Phase 0

- 자료 수집및 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

1.3.2. Phase 1 (03.20 ~ 03.28 **8**일)

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

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

표 3 Phase 1

- 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

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

1.3.3. Phase 2 (03.29 ~ 04.11 **10**일)

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

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 (**7**일)									
CI/CD	Argo CD 구현 및 Jenkins pipeline 구축(**7**일)									
Monitor	Prometheus, Grafana, Thanos, Cloud watch 구축(**7**일)									
보고서								기술문서, PPT 제작(**3**일)		

표 4 Phase 2

- Service 구현(Backend API, Frontend Funtion)
- CI/CD Pipeline 구현 (코드 관련 Jenkins, EKS 관련 argoCD)
- Monitoring Service 구현
- 고가용성 구현 (Kubemonkey, Prometheus HA)
- 작업 프로젝트 기술 문서, 발표 PPT, 영상작업

1.3.4. Phase 0~2 **Gantt chart**

단계 / 일자	03.13	03.14	03.15	03.16	03.17	03.20	03.21	03.22	03.23	03.24	03.27	03.28	03.29	03.30	03.31	04.03	04.04	04.05	04.06	04.07	04.10	04.11
자료 수집	5일																					
서비스						15일																
인프라						8일																
CI/CD													7일									
Monitor													7일									
보고서																				3일		

표 5 Gantt chart

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

1.4. 프로젝트 환경

활용 라이브러리 및 프레임워크

1.4.1. Service ENV

1.4.1.1. Database

Name	Version	Usage
MySQL (centos 7)	5.7	localhost에서 개발및 테스트 용도
MySQL 5.7 (RDS)	5.7	인프라 구축 완료 이후 사용 및 실제 서비스시 이용

표 6 Database env

1.4.1.2. Backend

Name	Category	Version	Usage
Java	language	11	java programing language
Spring boot	framework	2.5.9	java web framework
gradle	build tool	2.0.6	Groovy 기반 java build tool
jjwt	library	0.11.5	Java Json Web Token 암호화 및 인증
Hibernate	framework	5.3	JPA Hibernate 객체 관계 매핑
SCryptPasswordEncoder	class		Spring Security의 Password Encoder 중 하나 비밀번호 생성및 확인용

표 7 Backend env

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

1.4.1.3. Frontend

Name	Category	Version	Usage
Node.js	js runtime	18.15.0	node.js 빌드 및 실행
npm	package	9.5.0	node 패키지 관리자
Vue.js	framework	3.2.47	Groovy 기반 java build tool
vue cli	cli	5.0.8	Java Json Web Token 암호화 및 인증
Vite	build tool	2.9.8	빌드 도구
pinia	framework	2.0.14	스토어 라이브러리 및 상태관리 프레임워크
axios	http client	1.3.4	rest api 요청용
vee-validate	module	4.5.11	유효성 검사 모듈
bootstrap	framework	5.2.3	css 프레임워크

표 8 Frontend env

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

1.4.2. Infrastructure ENV

Name	Category	Version	Usage
Helm	Kubernetes Package Manager	3.11.1	쿠버네티스 패키지 관리자, EKS 클러스터 어플리케이션 배포 및 관리
AWS EKS	Kubernetes Manage Service	1.24	AWS 쿠버네티스 관리 도구
AWS RDS	Database	-	MSQL 8.0 실행
AWS EFS	File Storage	4.0	파일 기반 공유 스토리지
AWS Route53	Domain Service	-	도메인 이름 등록
Docker	Container Service	20.10.22	컨테이너 이미지 빌드

표 9 Infrastructure ENV

1.4.3. CI/CD ENV

Name	Category	Version	Usage
Jenkins	CI/CD	2.387.1	CI/CD 파이프라인 구성 및 빌드
ArgoCD	Deploy	v2.7.0-rc1	EKS 클러스터 배포
github	Code Manage	2.39.2.1	Jenkinsfile, Dockerfile 관리 및 EKS 오브젝트 관리
Docker Hub	Image Repository	-	Image 관리

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

1.4.4. Monitoring ENV

Name	Category	Version	Usage
Prometheus	Monitoring Tool	2.42.0	시계열 데이터로 metrics 수집
Grafana	Monitoring Tool	9.3.8	Prometheus의 메트릭 자료 시각화
Thanos	Monitoring & Logging	0.30.2	Prometheus의 고가용성 확보 & 모니터링 데이터 장기 저장
MinIO	Object Storage	5.0.7	Prometheus의 Object Storage
AWS CloudWatch	Monitoring Tool	-	AWS 리소스 모니터링

표 10 Monitoring ENV

1.4.5. 프로젝트 관리 방안

- 폭포수(Waterfall) 방식의 프로젝트 관리 방법론을 적용하여 프로젝트를 진행한다.

- 팀원들의 원활한 커뮤니케이션을 위해 협업툴을 적극 사용하고 개발 문서, 위키를 적도록 해서 효율적인 커뮤니케이션을 유지한다.

- Jenkins와 ArgoCD를 이용한 CI/CD 파이프라인을 통해 소스 코드를 지속적으로 빌드, 테스트하고, 배포하는 작업을

자동화하여 개발서버, 베타서버, 스테이징서버, 운영서버 등으로 나누어 지속적으로 배포를 할 수 있도록 지원한다.

1.5. 프로젝트 기대효과

1.5.1. 현황 (AS-IS)

- 동시에 수강신청을 하거나, 수업 정보를 조회하는 등의 작업으로 인해 서버 부하 증가 및 서버 지연의 문제가 발생함.
- 수업 정보 조회에 많은 시간이 걸림.
- 수강신청이 열리기를 기다리는데 시간을 다른 사이트에서 봐야 함.
- 학생들이 조회를 위해서 지속적인 새로고침을 하여 서버 조회가 다중 요청되어 서버가 지연됨.
- 수강신청 기간에 동시에 수강신청을 하거나, 수업 정보를 조회하는 등의 작업으로 인해 서버 부하증가 및 서버 지연으로 시스템 자체가 다운되거나 느려지는 등의 문제가 발생함.

1.5.2. 개선 및 기대효과 (TO-BE)

- 대규모 트래픽을 처리할 수 있는 분산 시스템을 구축하여 수강신청 시스템의 확장성 및 가용성을 높임.
- 다수의 백엔드에서의 로드밸런싱과 데이터베이스 성능 최적화및 캐시 서버를 도입하여 빠른 데이터 접근을 지원함.
- 수강신청 페이지 안에서 수강신청 시간까지 남은 시간을 표시하여 사용자의 불편함을 개선함.
- 프론트엔드의 반응형 기능으로 새로고침을 하지 않아도 지속적인 서비스 이용이 가능하게 되어 시스템의 부하를 줄임.

- 수강신청 기간, 수강신청 아닌기간에 맞춰 서버를 오토스케일링하여 원활한 서비스 이용 및 시스템 자원을 절약함.

2. 프로젝트 내용

2.1. Service

2.1.1. 서비스 기획 및 설계

2.1.1.1. 배경 및 목표

앞서 팀회의로 대학 수업신청 주제와 기대 효과에 맞는 서비스를 목표로 구현하는것을 목표로 정하며, 프로젝트의 서비스의 목적에 맞는 프로그래밍 언어및 프레임워크를 정하고 그것에 맞춰서 개발한다.

2.1.1.2. 서비스 진행 프로세스

1) 서비스 기획
2) 데이터베이스 설계
3) 백엔드 설계
4) 프론트엔드 설계
5) 인프라팀에 필요 인프라 전달
6) 실제 구현 단계

2.1.1.3. 서비스 기획(기능정리)

다른 대학교 수강신청들의 서비스들을 비교 및 분석하여 필요한 기능을 정리한다.

- ❖ 수강 신청 기능
- ❖ 로그인기능
- ❖ 내 정보 수정 기능
- ❖ 회원가입 기능

- ❖ 장바구니 기능
- ❖ 시간에 맞춰 수강 신청이 열리는 기능
- ❖ 강의 생성 기능
- ❖ 위 기능들에 관련한 정보를 제공하는 기능

2.1.1.4. 서비스 기획(Use case)

앞서 정리한 기능을 최적화 하여 기능을 구현하는데 있어서 ADMIN 과 USER 의 두 사용자 부류로 나누어서 서비스를 기획하게 되었다.

ADMIN	대학교 교직원, 수강신청에 관한 권한및 학생을 관리할 수 있는 권한을 가진 사람
USER	대학교 학생, 강의를 들을 수 있는 권한을 가진 사람

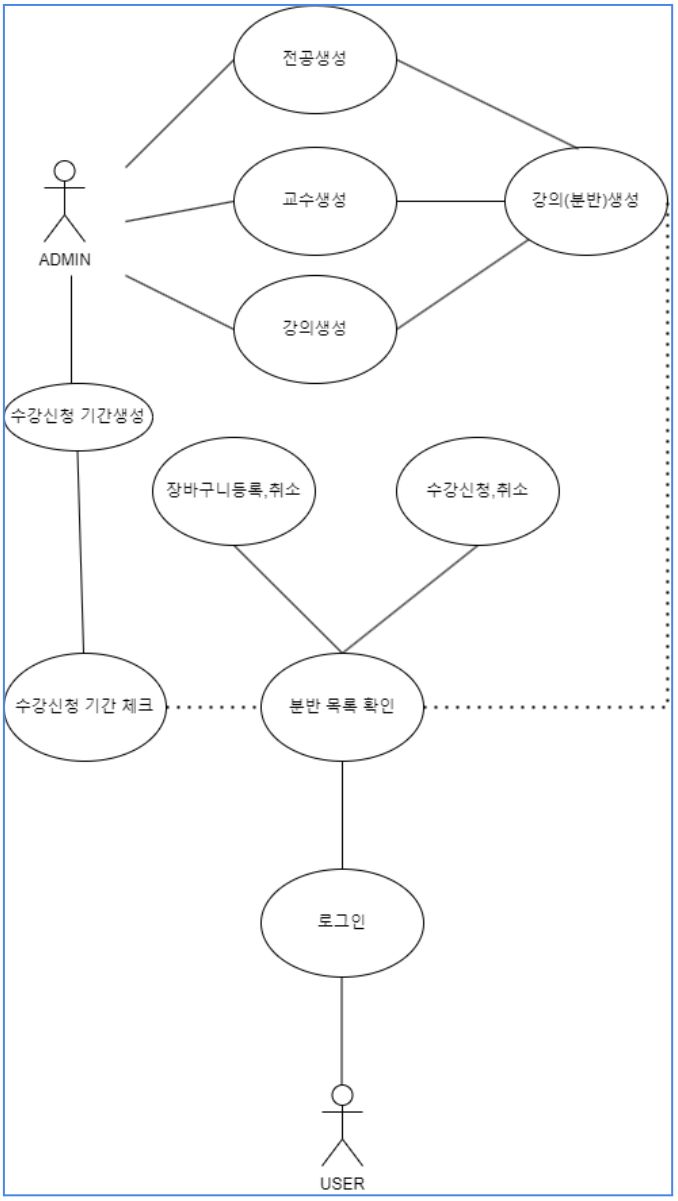


그림 1 service usecase

2.1.2. DB

2.1.2.1. 데이터베이스 환경

데이터베이스 소프트웨어는 MySQL로 정하여 진행하게 되었다.
이 프로젝트에서 MySQL으로 진행한 이유는 현재 구현중인 수강 신청 시스템과 관련해서 관계형 데이터베이스 가 데이터베이스를 표현하고 구현하는 데에 효율적이라고 생각했고, 수많은 관계형 데이터베이스들 중에서 오픈소스

라이센스로써 무료로 사용 가능하고 다양한 운영체제에서 사용할 수 있고 많은 사용자를 가지고 있는 MySQL을 선택하였다. 그리고 추후 기획이 바뀔 시에 다른 데이터베이스를 사용하게 되게 된다면 마이그레이션 시에 레퍼런스 자료가 많다는 점도 장점으로 생각하여 MySQL로 작업을 하기로 하였다.

또한 고려할 요소들 중 하나는 버전인데, MySQL 8.0과 5.7의 버전을 비교해보면서 둘중 하나로 정하려고 했었고, 8.0의 장점이 많았지만 프로젝트 개발자가 8.0의 사용 경험이 없어 혹시 모를 이슈에 대처하기 어려울 것 같은 점, 5.7은 속도가 느리다는 단점이 있지만 인터넷 상에서 찾을 수 있는 레퍼런스가 많다는 장점이 있어서 비교적 최근에 나온 MySQL 8.0보다 5.7 버전으로 프로젝트 DB를 구성하기로 하였다.

MySQL 5.7 (centos7)	localhost에서 개발및 테스트 용도
MySQL 5.7 (RDS)	인프라 구축 완료 이후 사용 및 실제 서비스시 이용

표 11 프로젝트에서 사용한 DB system

2.1.2.2. 데이터베이스 구성

앞서 기획한 **UseCase** 를 기반으로 데이터베이스를 구성하였다.

그림 2 DB diagram

https://dbdiagram.io/d/641d702b5758ac5f1723d69e

2.1.2.2.1. Admin Table

Admin 의 정보를 담은 테이블

그림 3 Admin Table

Column	experience
admin_id	기본키
admin_admin_id	로그인시의 id로 사용
admin_name	admin의 이름
admin_password	비밀번호 , 로그인시의 pw

표 12 Admin Table

2.1.2.2.2. User Table

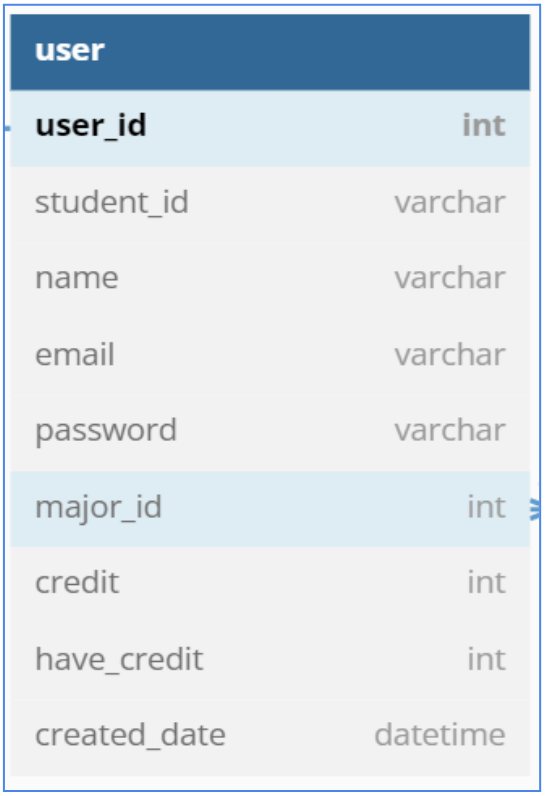


그림 4 User Table

Column	experience
user_id	기본키 | take_lecture, cart_lecture 와 다대일 관계
student_id	로그인시의 id로 사용 , User 학번
name	User의 이름 , 학생 이름
email	이메일
password	비밀번호 , 로그인시의 pw
major_id	전공ID | major.major_id 와 일대다 관계
credit	최대 수강 가능 학점
have_credit	현재 수강한 강의의 학점 합
created_date	가입한, 생성된 날짜 및 시간

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

표 13 User Table

2.1.2.2.3. Major Table

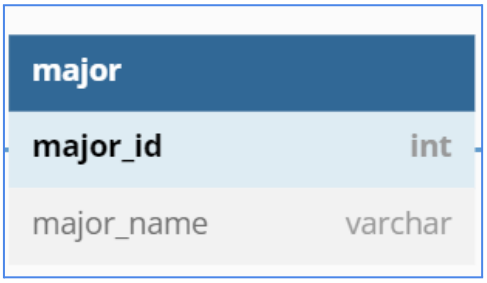


그림 5 Major Table

Column	experience
major_id	기본키 | user.major_id, professor_major_id,lecture_major_id 다대일 관계
major_name	전공 이름

표 14 Major Table

2.1.2.2.4. Professor Table

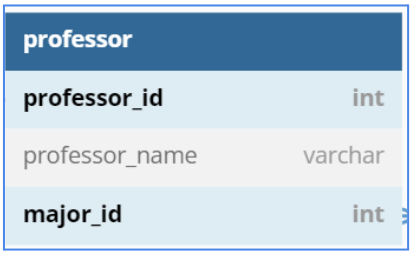


그림 6 Professor Table

Column	experience
professor_id	기본키 | lecture_class.professor_id 다대일 관계
professor_name	전공 이름
major_id	전공ID | major.major_id 와 일대다 관계

표 15 Professor Table

2.1.2.2.5. Lecture Table

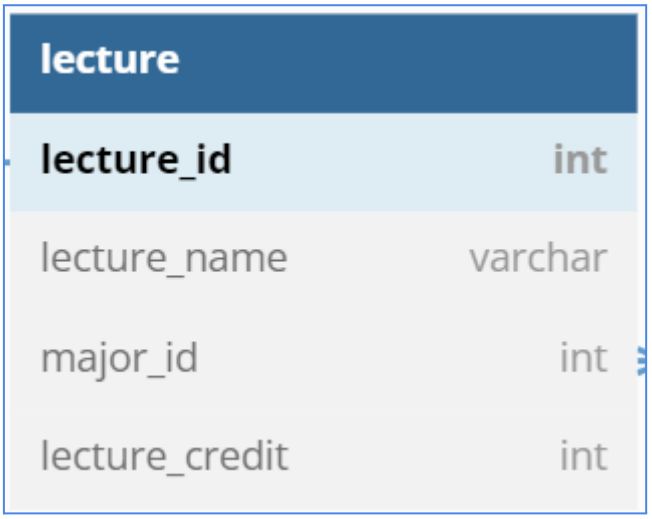


그림 7 Lecture Table

Column	experience
lecture_id	기본키 | lecture_class.lecture_id 다대일 관계
lecture_name	강의 이름
major_id	전공ID | major.major_id 와 일대다 관계
lecture_credit	강의 학점

표 16 Lecture Table

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.1.2.2.6. Lecture_Class Table

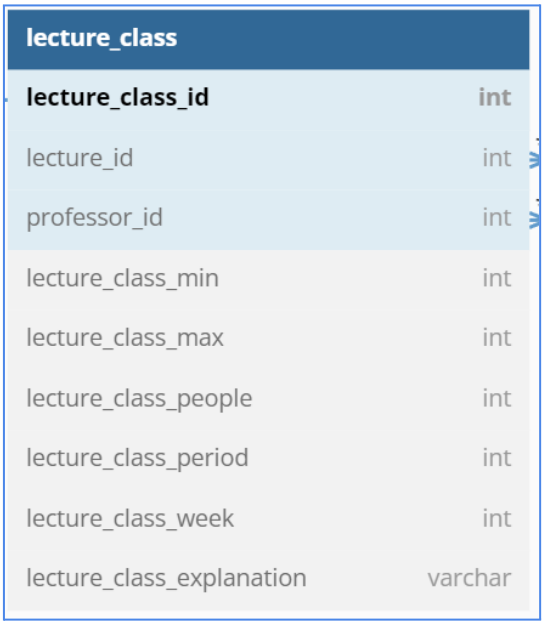


그림 8 Lecture_Class Table

Column	experience
lecture_class_id	기본키 | cart_lecture , take_lecture 다대일 관계
lecture_id	강의ID | lecture.lecture_id 와 일대다 관계
professor_id	교수ID | major.major_id 와 일대다 관계
lecture_class_min	수강 시작 최소 인원
lecture_class_max	수강 가능 최대 인원
lecture_class_people	현재 수강 인원
lecture_class_period	수업 교시
lecture_class_week	수업 요일
lecture_class_enplanation	강의 설명

표 17 Lecture_Class Table

2.1.2.2.7. Take_Lecture Table

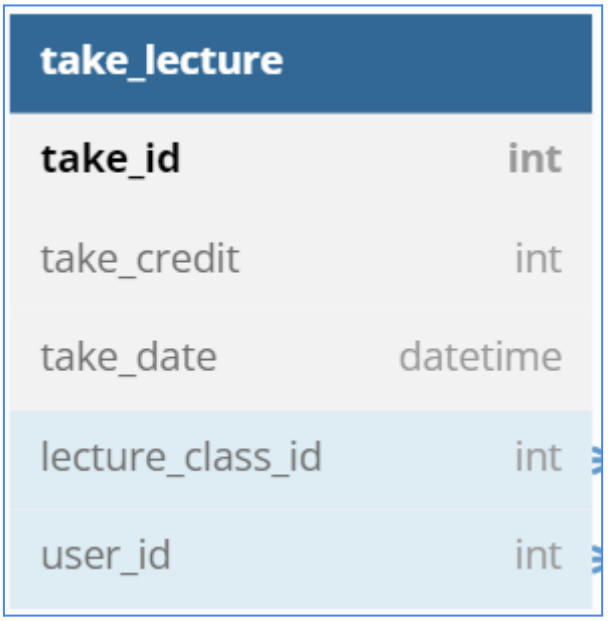


그림 9 Take_Lecture Table

Column	experience
take_id	기본키 | cart_lecture , take_lecture 다대일 관계
take_credit	강의의 학점
take_date	수강신청 신청한 날짜와 시간
lecture_class_id	강의ID | lecture_class.lecture_class_id 와 일대다 관계
user_id	수강한 학생 | id user.user_id 와 일대다 관계

표 18 Take_Lecture Table

2.1.2.2.8. Cart_Lecture Table

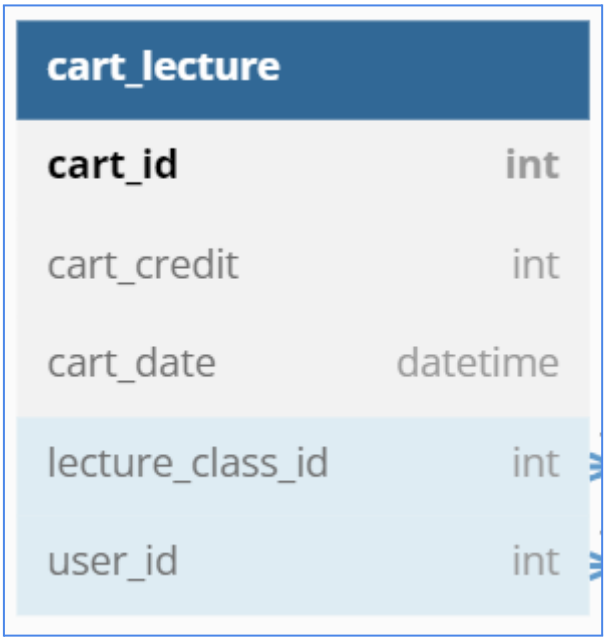


그림 10 Cart_Lecture Table

Column	experience
cart_id	기본키 | cart_lecture , take_lecture 다대일 관계
cart_credit	강의의 학점
cart_date	장바구니 등록한 날짜와 시간
lecture_class_id	강의ID | lecture_class.lecture_class_id 와 일대다 관계
user_id	수강한 학생 | id user.user_id 와 일대다 관계

표 19 Cart_Lecture Table

	Open			G3M
	Category	첨부파일 버전	문서 최종 수정일	
	Manual + Utility	1.1	2023.04.07	

2.1.2.2.9. Open_Time Table

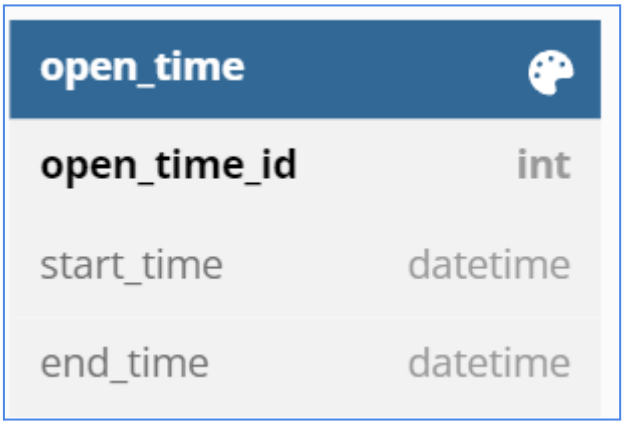


그림 11 Open_Time Table

Column	experience
open_time_id	기본키
start_time	수강신청 시작 시간
ent_time	수강신청 종료 시간

표 20 Open_Time Table

2.1.2.3. 데이터베이스 코드

MySQL

```
CREATE TABLE `user` (
  `user_id` int PRIMARY KEY,
  `student_id` varchar(255),
  `name` varchar(255),
  `email` varchar(255),
  `password` varchar(255),
  `major_id` int,
  `credit` int,
  `have_credit` int,
  `created_date` datetime
);
CREATE TABLE `major` (
  `major_id` int PRIMARY KEY,
  `major_name` varchar(255)
);
CREATE TABLE `professor` (
  `professor_id` int,
  `professor_name` varchar(255),
  `major_id` int,
  PRIMARY KEY (`professor_id`, `major_id`)
);
CREATE TABLE `lecture` (
  `lecture_id` int PRIMARY KEY,
  `lecture_name` varchar(255),
  `major_id` int,
  `lecture_credit` int
);
CREATE TABLE `lecture_class` (
  `lecture_class_id` int PRIMARY KEY,
  `lecture_id` int,
  `professor_id` int,
  `lecture_class_min` int,
  `lecture_class_max` int,
  `lecture_class_people` int,
  `lecture_class_period` int,
  `lecture_class_week` int,
```

```
  `lecture_class_explanation` varchar(255)
);
CREATE TABLE `take_lecture` (
  `take_id` int PRIMARY KEY,
  `take_credit` int,
  `take_date` datetime,
  `lecture_class_id` int,
  `user_id` int
);
CREATE TABLE `admin` (
  `admin_id` int PRIMARY KEY,
  `admin_admin_id` varchar(255),
  `admin_name` varchar(255),
  `admin_password` varchar(255)
);
CREATE TABLE `open_time` (
  `open_time_id` int PRIMARY KEY,
  `start_time` datetime,
  `end_time` datetime
);
CREATE TABLE `cart_lecture` (
  `cart_id` int PRIMARY KEY,
  `cart_credit` int,
  `cart_date` datetime,
  `lecture_class_id` int,
  `user_id` int
);
ALTER TABLE `user` ADD FOREIGN KEY (`major_id`) REFERENCES `major`
(`major_id`);
ALTER TABLE `professor` ADD FOREIGN KEY (`major_id`) REFERENCES `major`
(`major_id`);
ALTER TABLE `lecture` ADD FOREIGN KEY (`major_id`) REFERENCES `major`
(`major_id`);
ALTER TABLE `lecture_class` ADD FOREIGN KEY (`lecture_id`) REFERENCES
`lecture` (`lecture_id`);
ALTER TABLE `lecture_class` ADD FOREIGN KEY (`professor_id`) REFERENCES
`professor` (`professor_id`);
ALTER TABLE `take_lecture` ADD FOREIGN KEY (`user_id`) REFERENCES `user`
(`user_id`);
ALTER TABLE `take_lecture` ADD FOREIGN KEY (`lecture_class_id`)
```

```
REFERENCES `lecture_class` (`lecture_class_id`);
ALTER TABLE `cart_lecture` ADD FOREIGN KEY (`user_id`) REFERENCES `user`
(`user_id`);
ALTER TABLE `cart_lecture` ADD FOREIGN KEY (`lecture_class_id`)
REFERENCES `lecture_class` (`lecture_class_id`);
```

코드 service_init_mysql.spl

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.1.3. Backend

2.1.3.1. Backend 환경

이 프로젝트에서 Spring Boot 프레임워크로 진행하게 되었다
Spring Boot를 사용한 이유로는 으로 가장 무난하며 많이 사용되는Spring Boot를 통하여 Spring 프레임워크를 경험을 해보며 jenkins로 java빌드하여 CI를 프로젝트에 보여줄 수 있다는 장점이있어서이다.
또한 프로젝트의 시간적 여유가 생겨서 간단한 모델을 적용한 Python deep learning을 이용해 POST로 전달받아 결과값을 알려주는 Flask백엔드를 구현하여 요즘 많은 기업에서 채택하고 있는 방식인 MSA 방식의 개발방법을 어느정도 맛보기로 개발하게 되었다.

Name	Category	Version	Usage
Java	language	11	java programing language
Spring boot	framework	2.5.9	java web framework
grable	build tool	2.0.6	Groovy 기반 java build tool
jjwt	library	0.11.5	Java Json Web Token 암호화 및 인증
Hibernate	framework	5.3	JPA Hibernate 객체 관계 매핑
SCryptPasswordEncoder	class		Spring Security의 Password Encoder 중 하나 비밀번호 생성및 확인용

표 21 Backend 환경

2.1.3.2. Backend 구성

Spring Boot프레임워크로 개발을 진행하면서 앞서 기획에서의 기능을 보고 구현한 DB와 맞게 API를 설계한다.
MVC 패턴으로 기획한 설계의 기능에 맞는 **REST API**를 구현하는것을 목표로 구성되었다.

2.1.3.3. Backend (Spring)

스프링으로 구현한 수강신청어플리케이션의 REST API이다.
https://github.com/tkfka1/course-registration-GoormUniversity-restapi

2.1.3.3.1. Web Layer

컨트롤러(@Controller)가 대표적이고, 이외에도 필터(@filter), 인터셉터, 컨트롤러 어드바이스 등이 포함된다.
외부 요청과 응답에 대한 전반적인 영역을 의미한다.

소스	설명
AdminController.java	교직원의 데이터
CartLectureController.java	교직원의 데이터 수정
LectureClassController.java	장바구니의 데이터
LectureController.java	장바구니의 데이터 수정
MajorController.java	강의의 데이터
OpenTimeController.java	강의의 데이터 수정
ProfessorController.java	강의 분반의 데이터 수정
TakeLectureController.java	강의 분반의 데이터 수정
TakeLectureController.java	교수의 데이터

표 22 Web layer

2.1.3.3.2. Service Layer

말 그대로 서비스(@Service)이다.
일반적으로 컨트롤러와 저장소(Repository, Dao)의 중간에 위치한다.
트랜잭션(@Transactional)과 도메인 간의 연산 순서를 보장해 준다.

소스	설명
AdminService.java	교직원 서비스
CartLectureService.java	장바구니 서비스
LectureClassService.java	강의분반 서비스
LectureService.java	강의 서비스
MajorService.java	전공 서비스
OpenTimeService.java	강의 시작 시간 서비스
ProfessorService.java	교수 서비스
TakeLectureService.java	수강신청 서비스
UserService.java	학생 서비스

표 23 Web layer

2.1.3.3.3. Repository Layer

DB와 같은 데이터 저장소에 접근하는 영역이다.
JPA를 사용하였다.

소스	설명
AdminRepository.java	교직원 레포지토리
CartLectureRepository.java	장바구니 레포지토리
LectureClassRepository.java	강의분반 레포지토리
LectureRepository.java	강의 레포지토리
MajorRepository.java	전공 레포지토리
OpenTimeRepository.java	수강신청 시간 레포지토리
ProfessorRepository.java	교수 레포지토리
TakeLectureRepository.java	수강신청 레포지토리
UserRepository.java	학생 레포지토리

표 24 repository class

```
package com.spring.api.repository;

import com.spring.api.domain.User;
import org.springframework.data.repository.CrudRepository;

import java.util.Optional;

public interface UserRepository extends CrudRepository<User, Long> ,
UserRepositoryCustom{

    Optional<User> findByStudentId(String studentId);

}
```

코드 UserRepository.java

2.1.3.3.4. DTOs

DTO(Data Transfer Object)는 계층 간의 데이터 교환을 위한 객체를 이야기한다.

소스	설명
AdminEdit.java	교직원 수정 객체
AdminLogin.java	교직원의 로그인 객체
AdminSignup.java	교직원 등록 객체
CartLectureEdit.java	장비구니 수정 객체
CartLectureSignup.java	장바구니 등록 객체
LectureClassEdit.java	강의 분반 수정 객체
LectureClassSignup.java	강의 분반 등록 객체
LectureEdit.java	강의 수정 객체
LectureSignup.java	강의 등록 객체
MajorEdit.java	전공 수정 객체
MajorSignup.java	전공 등록 객체
OpenTimeEdit.java	수강신청 오픈 시간 수정 객체
OpenTimeSignup.java	수강신청 오픈 시간 등록 객체
ProfessorEdit.java	교수 수정 객체
ProfessorSignup.java	교수 등록 객체
TakeLectureEdit.java	수강신청 수정 객체
TakeLectureSignup.java	수강신청 등록 객체
UserEdit.java	학생 수정 객체
UserLogin.java	학생 로그인 객체
UserSignup.java	학생 가입 객체

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.1.3.3.5. Domain model

개발 대상, 즉 도메인을 모든 사람이 동일한 관점에서 이해할 수 있고 공유할 수 있도록 단순화한 것, 비즈니스 로직을 처리하는 영역이다.
JPA를 사용한다면, @Entity가 사용되는 영역 역시 도메인 영역이다

소스	설명
Admin.java	교직원의 데이터
AdminEditor.java	교직원의 데이터 수정
CartLecture.java	장바구니의 데이터
CartLectureEditor.java	장바구니의 데이터 수정
Lecture.java	강의의 데이터
LectureEditor.java	강의의 데이터 수정
LectureClass.java	강의 분반의 데이터 수정
LectureClassEditor.java	강의 분반의 데이터 수정
Major.java	교수의 데이터
MajorEditor.java	교수의 데이터 수정
OpenTime.java	수강신청 오픈 시간교 데이터
OpenTimeEditor.java	수강신청 오픈 시간교 데이터 수정
Professor.java	교수의 데이터
ProfessorEditor.java	교수의 데이터 수정
TakeLecture.java	수강신청의 목록 데이터
TakeLectureEditor.java	수강신청의 목록 데이터 수정
User.java	학생의 데이터
UserEditor.java	학생의 데이터 수정

admin.java 하나의 코드를 살펴본다

```
package com.spring.api.domain;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonManagedReference;
import lombok.AccessLevel;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Getter
@Entity
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class Admin {

## 컬럼에 데이터를 배치
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "admin_id")
    private Long id;
    @Column(name = "admin_admin_id")
    private String adminId;
    @Column(name = "admin_name")
    private String name;
    @Column(name = "admin_password")
    private String password;

## 데이터 Getter를 위한 builder
    @Builder
    public Admin(String adminId, String name, String password) {
        this.adminId = adminId;
        this.name = name;
        this.password = password;
    }

## 일대다로 세션어드민과 연동
```

```
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "admin" , fetch =
FetchType.LAZY)
    @JsonIgnore
    private List<SessionAdmin> sessions = new ArrayList<>();

    public SessionAdmin addSession() {
        SessionAdmin session = SessionAdmin.builder()
                .admin(this)
                .build();
        sessions.add(session);

        return session;
    }

## 도메인의 교직원 에디터와 상호작용
    public AdminEditor.AdminEditorBuilder toEditor() {
        return AdminEditor.builder()
                .adminId(adminId)
                .name(name)
                .password(password);
    }

    public void edit(AdminEditor userEditor) {
        adminId = userEditor.getAdminId();
        name = userEditor.getName();
        password = userEditor.getPassword();
    }

}
```

코드 admin.java

2.1.3.3. Backend (Flask)

플라스크로 구현한 deep learning 모델을 연산해서 반환하는 REST API이다.
https://github.com/tkfka1/course-registration-GoormUniversity-animalapi
간단하게 구현한만큼 그냥 단일 파일 하나로 작성하게 되었다.
densenet121 모델을 사용했고, imagenet_class_index.json 에 키값을 전달받아서 동물의 이름을 출력하게 만들었다.

main.py

```
import io
from torchvision import models
import json
from flask import Flask, request
from flask import make_response
import torchvision.transforms as transforms
from PIL import Image

app = Flask(__name__)

## 이미지를 받아오기
def transform_image(image_bytes):
    my_transforms = transforms.Compose([transforms.Resize(255),
                                        transforms.CenterCrop(224),
                                        transforms.ToTensor(),
                                        transforms.Normalize(
                                            [0.485, 0.456, 0.406],
                                            [0.229, 0.224, 0.225])])
    image = Image.open(io.BytesIO(image_bytes))
    return my_transforms(image).unsqueeze(0)
## 모델 가져오기
model = models.densenet121(pretrained=True)
model.eval()
## 데이터 인덱스
imagenet_class_index = json.load(open('imagenet_class_index.json'))

def get_prediction(image_bytes):
    tensor = transform_image(image_bytes=image_bytes)
    outputs = model.forward(tensor)
```

```
    _, y_hat = outputs.max(1)
    predicted_idx = str(y_hat.item())

    return imagenet_class_index[predicted_idx]

## 테스트 get 요청
@app.route('/', methods=['GET'])
def hell():
    if request.method == 'GET':
        print("GET request received")
        return "Hell World!"
ㅌ
@app.route('/mlapp/app', methods=['GET'])
def hello():
    if request.method == 'GET':
        print("GET request received")
        return "Hello World!"
## 동물판독기 POST 요청
@app.route('/mlapp/mlapp', methods=['POST'])
def predict():
    if request.method == 'POST':
        print("POST request received")
        file = request.files['image']
        img_bytes = file.read()

        class_id, class_name = get_prediction(image_bytes=img_bytes)

        res = {
            'class_id' : class_id,
            'class_name' : class_name
        }
        res = make_response(json.dumps(res, ensure_ascii=False))
        res.headers['Content-Type'] = 'application/json'

        return res

if __name__=="__main__":
    app.run(host="0.0.0.0",debug=True)
```

코드 main.py

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.1.3.4. Backend API 모음

(spring)

/api/admin/auth/login	POST	교직원 로그인
/api/admin/auth/signup	POST	교직원 회원가입
/api/admin/auth/{id}	GET, DELETE, PUT, PATCH	(ID 단일) 정보조회, 삭제, 수정
/api/admin/auth	GET	전체 교직원 조회
/api/user/auth/login	POST	학생 로그인
/api/user/auth/signup	POST	학생 회원가입
/api/user/auth/{id}	Get, DELETE, Put, PATCH	(ID 단일) 정보조회, 삭제, 수정
/api/user/auth	GET	전체 학생 조회
/api/lecture/auth/signup	POST	강의 생성
/api/lecture/auth/{id}	Get, DELETE, PUT, PATCH	(ID 단일) 정보조회, 삭제, 수정
/api/lecture/auth	GET	전체 강의 조회
/api/lectureClass/auth/signup	POST	분반 생성
/api/lectureClass/auth/{id}	Get, DELETE, PUT, PATCH	(ID 단일) 정보조회, 삭제, 수정
/api/lectureClass/auth	GET	전체 분반 조회
/api/take/auth/signup	POST	수강신청 생성
/api/take/auth/{id}	Get, DELETE, PUT, PATCH	(ID 단일) 정보조회, 삭제, 수정
/api/take/auth	GET	전체 수강신청 조회
/api/cart/auth/signup	POST	장바구니 생성

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

/api/cart/auth/{id}	Get, DELETE, PUT, PATCH	(ID 단일) 정보조회, 삭제, 수정
/api/cart/auth	GET	전체 장바구니 조회
/api/major/auth/signup	POST	전공 생성
/api/major/auth/{id}	Get, DELETE, PUT, PATCH	(ID 단일) 정보조회, 삭제, 수정
/api/major/auth	GET	전체 전공 조회
/api/professor/auth/signup	POST	교수 생성
/api/professor/auth/{id}	Get, DELETE, PUT, PATCH	(ID 단일) 정보조회, 삭제, 수정
/api/professor/auth	GET	전체 교수 조회
/api/time/auth/signup	POST	수강신청오픈시간 생성
/api/time/auth/{id}	Get, DELETE, PUT, PATCH	(ID 단일) 정보조회, 삭제, 수정
/api/time/auth	GET	전체 수강신청오픈시간 조회
/mlapp/mlapp	POST	AI 서비스 동물판독기

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.1.4. Frontend

2.1.4.1. Frontend 환경

이 프로젝트에서 Vue.js 프레임워크로 진행하게 되었다으로 Vue 를사용한 이유는 jenkins 를 사용하기 때문에 빌드를 하는 프론트엔드프론트엔드에서 선택을 하려고 했고, 제일 실무에서 많이 사용하는 Vue와 React 중에서 고민을 했다.
개발자는 두 프레임워크 사용경험이 없었고 Vue의 러닝커브가 상대적으로 짧고 Vue로도 충분히 좋은 프론트엔드를 구현할 수 있을거라는 생각을 하여 React와 Vue 중에서 Vue를 선택하게 되었다.

Name	Category	Version	Usage
Node.js	js runtime	18.15.0	node.js 빌드 및 실행
npm	pakage	9.5.0	node 패키지 관리자
Vue.js	framework	3.2.47	Groovy 기반 java build tool
vue cli	cli	5.0.8	Java Json Web Token 암호화 및 인증
Vite	build tool	2.9.8	빌드 도구
pinia	framework	2.0.14	스토어 라이브러리 및 상태관리 프레임워크
axios	http client	1.3.4	rest api 요청용
vee-validate	module	4.5.11	유효성 검사 모듈
bootstrap	framework	5.2.3	css 프레임워크

표 25 Frontend 도구

2.1.4.2. Frontend 구성

Vue.js는 node.js 기반의 컴포넌트 를 사용하여 작은 단위로 쪼개어서 개발을 할 수 있다는 장점이 있다.

전체적인 Frontend는 두 부분으로 나누어서 진행했는데
admin(교직원) 부분과 user(학생) 부분의 프론트엔드 두 부분으로 만들었다.

두 부분으로 나눈 이유는 처음 기획으로는 한번에 같이 개발을해서 <admin.domain> 방식으로 admin과 user 를 구분을 하는 것이였는데 로그인 인증시에 사용하는 토큰을 나누어 진행을 하는데 있어서 잦은 오류가 생겨서 두 부분으로 나누어서 개발을 진행하게 되었고.

앞서 구현한 백엔드의 API 설계에 맞게 구현을 하는것을 목표로
대부분의 기능은 백엔드의 REST API를 받아서 실제 정보를 주고받는 행위로 이루어진다.

2.1.4.3. Frontend (admin)

admin 부분의 프론트엔드이다

- 로그인

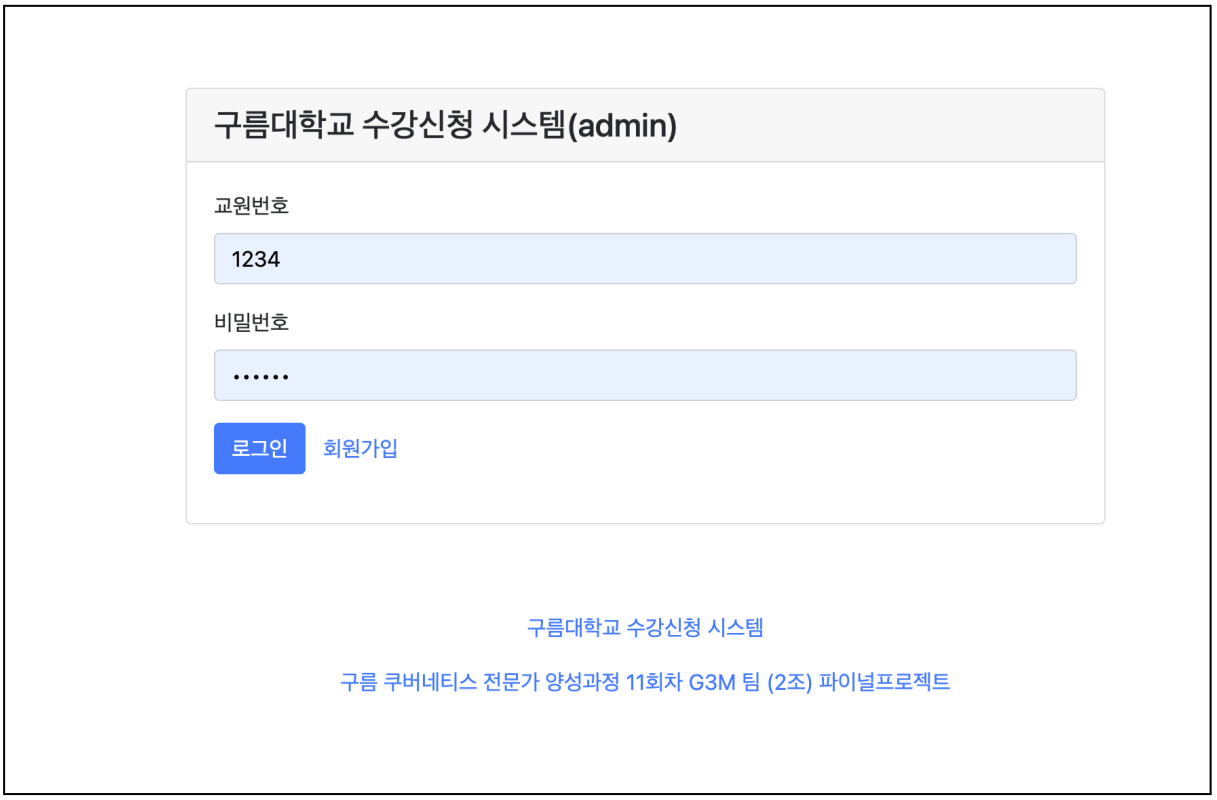


그림 12 로그인

- 회원가입

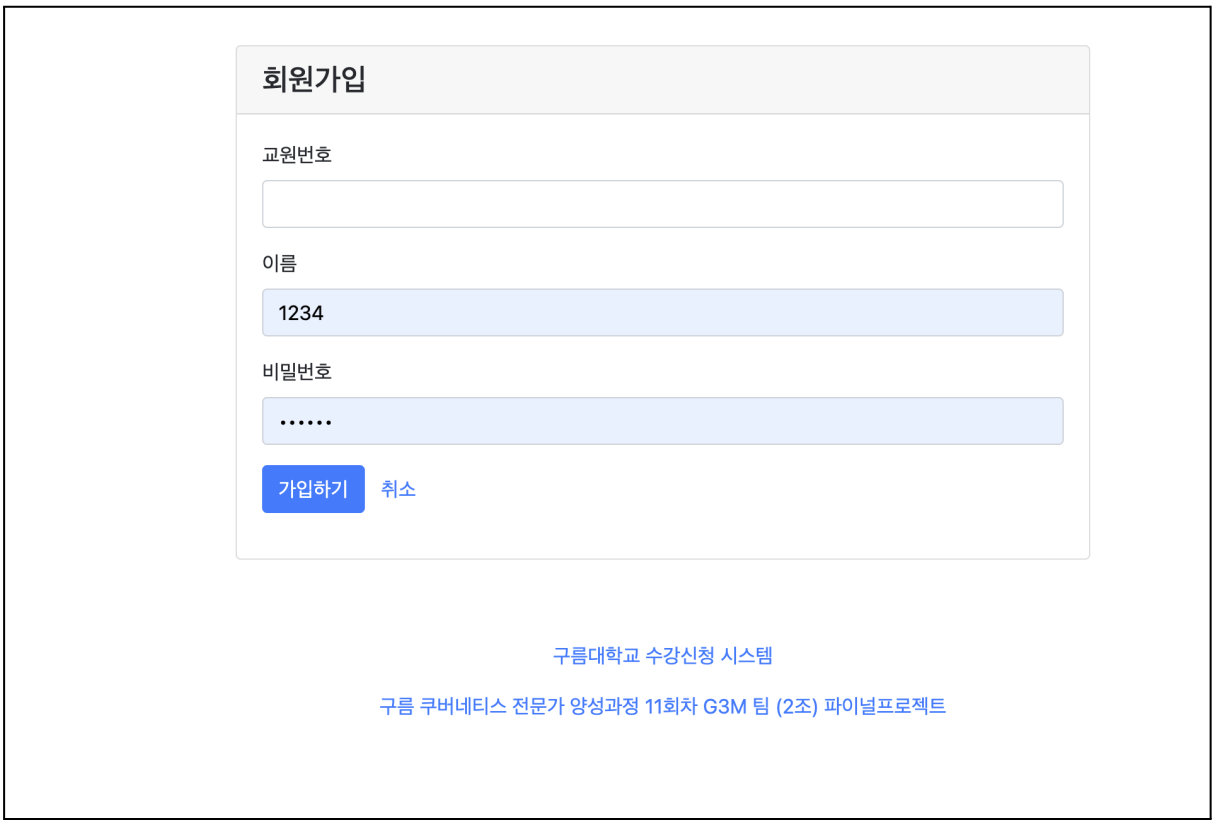


그림 13 회원가입

- 메인페이지

그림 14 메인페이지

- 교직원

그림 15 교직원

- 학생 - (학생세부 수강)

그림 16 학생

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

- 교수

그림 17 교수

- 전공

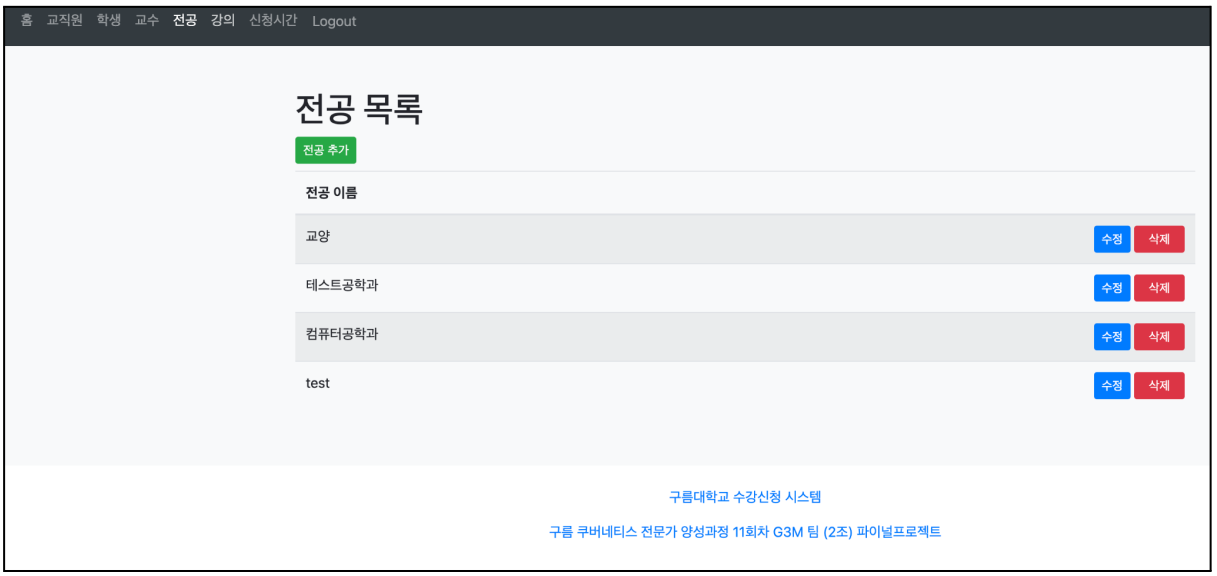


그림 18 전공

<table>
<tr><td rowspan="3"></td><td colspan="3">Open</td><td rowspan="3">G3M</td></tr>
<tr><td>Category</td><td>첨부파일 버전</td><td>문서 최종 수정일</td></tr>
<tr><td>Manual + Utility</td><td>1.1</td><td>2023.04.07</td></tr>
</table>

- 수업 - (수업세부 분반)

그림 19 수업

- 수강신청개시시간

그림 20 수강신청개시시간

2.1.4.4. Frontend (user)

user 부분의 프론트엔드이다
구성은 다음과 같다

- 로그인

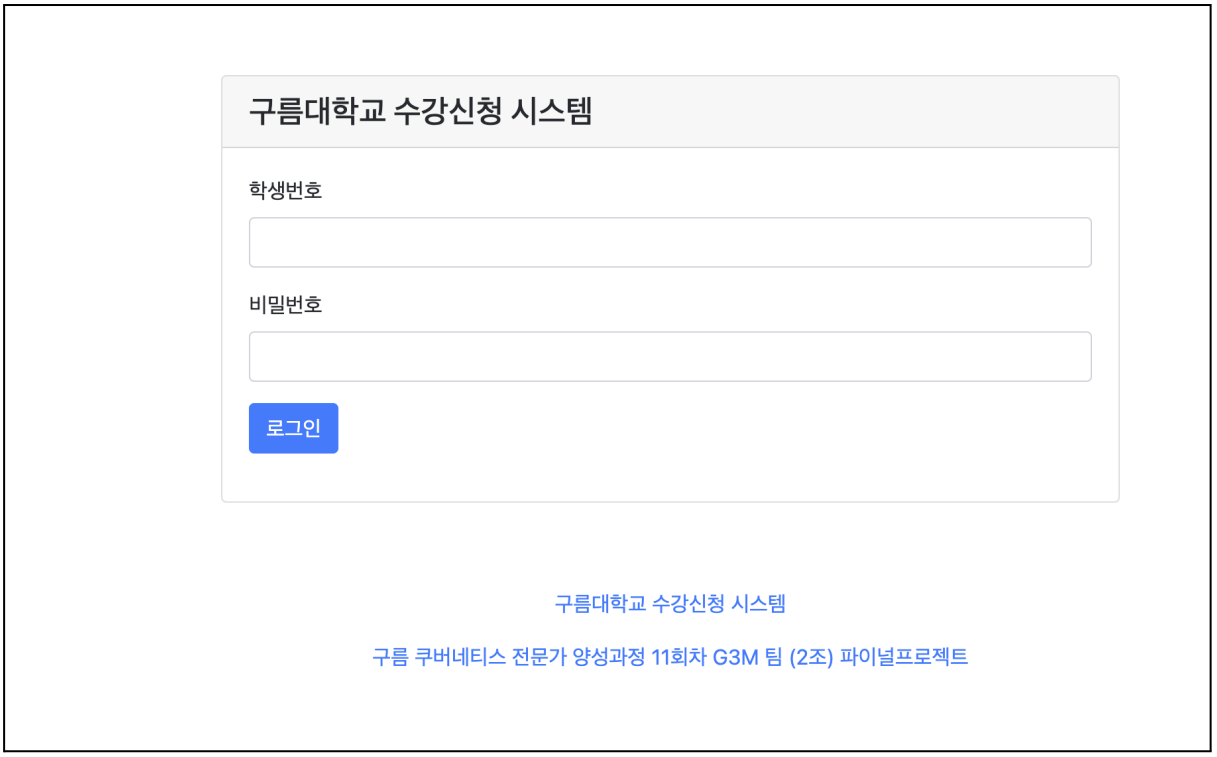


그림 21 User 로그인

- 메인페이지

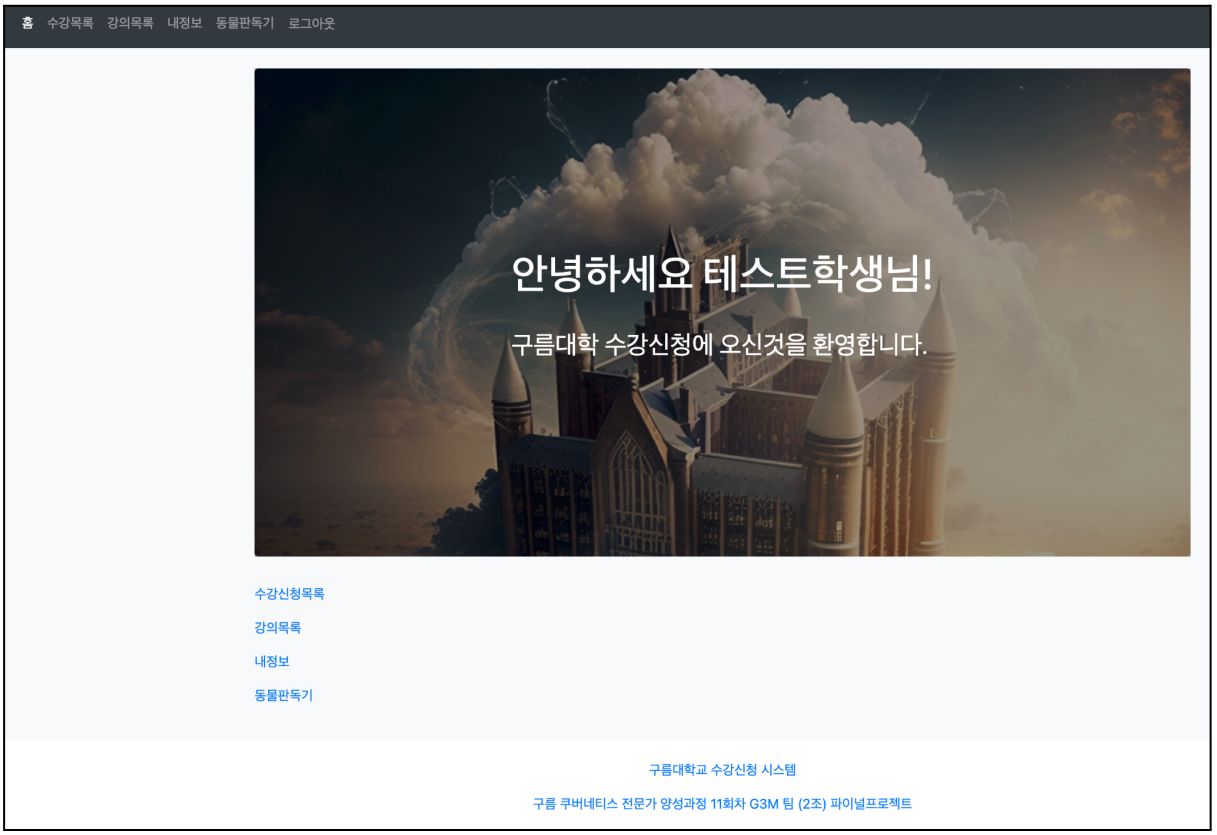


그림 22 User 메인페이지

- 마이페이지

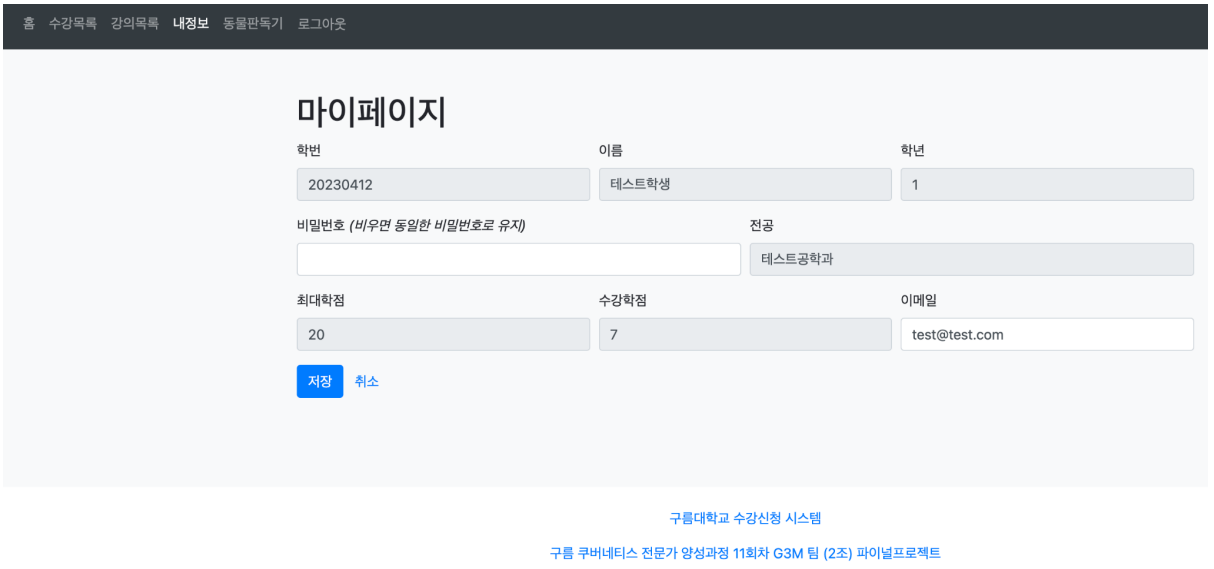


그림 23 User 마이페이지

- 장바구니 및 수강목록

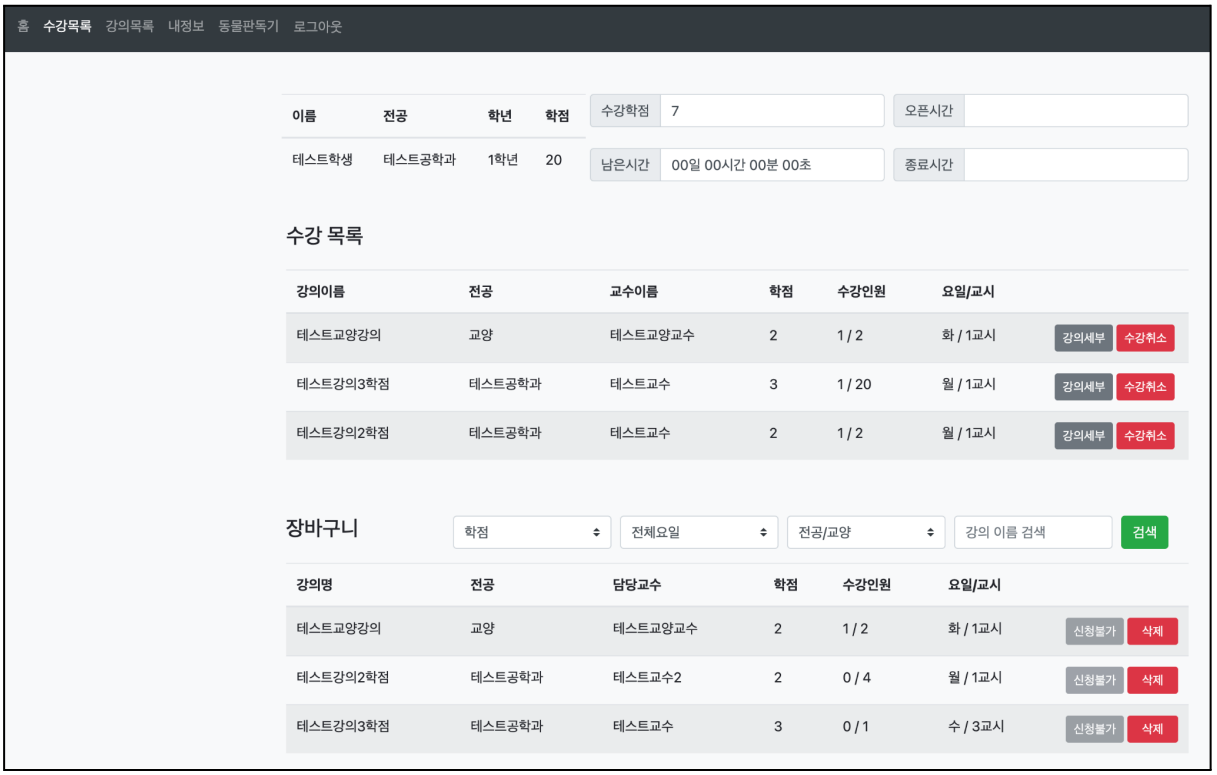


그림 24 User 장바구니 및 수강목록

- 개설 강의 목록

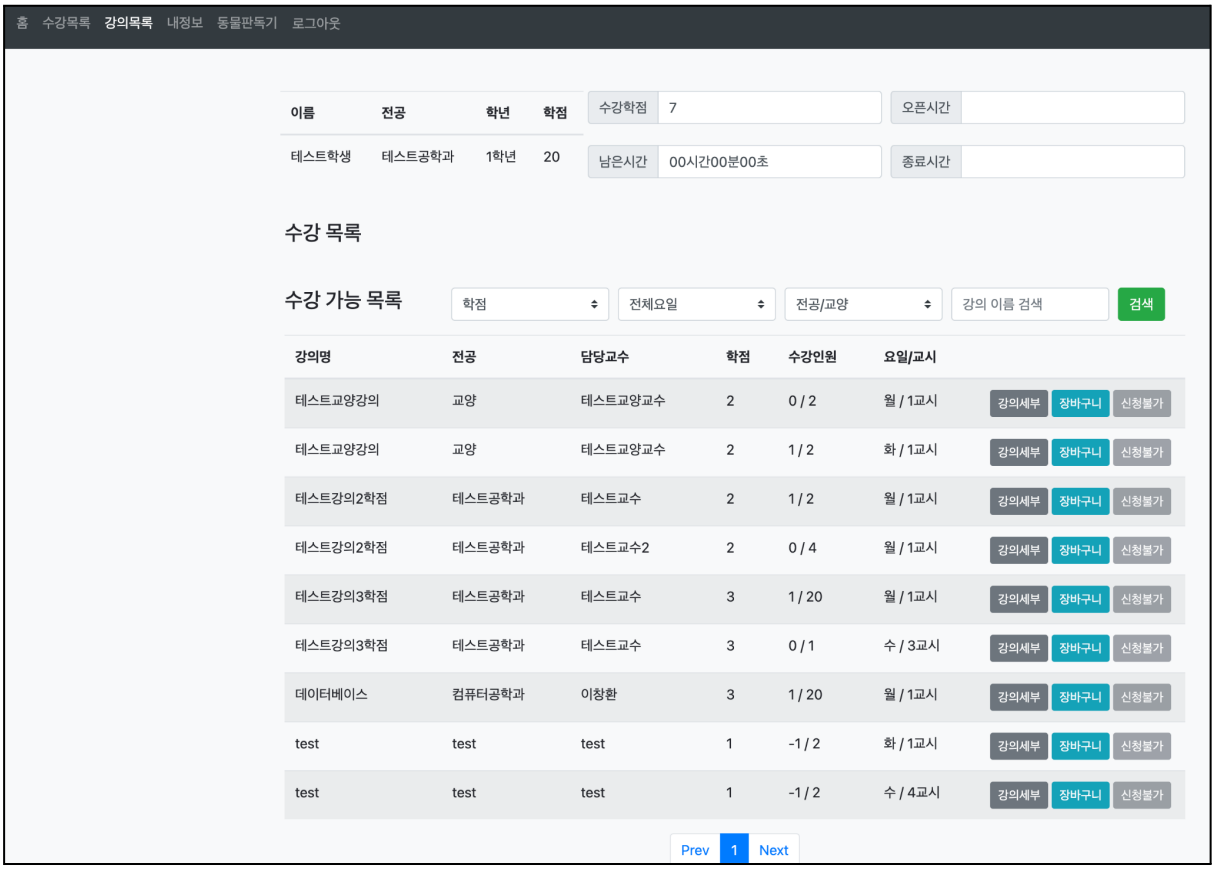


그림 25 User 강의 목록

- 동물판독기

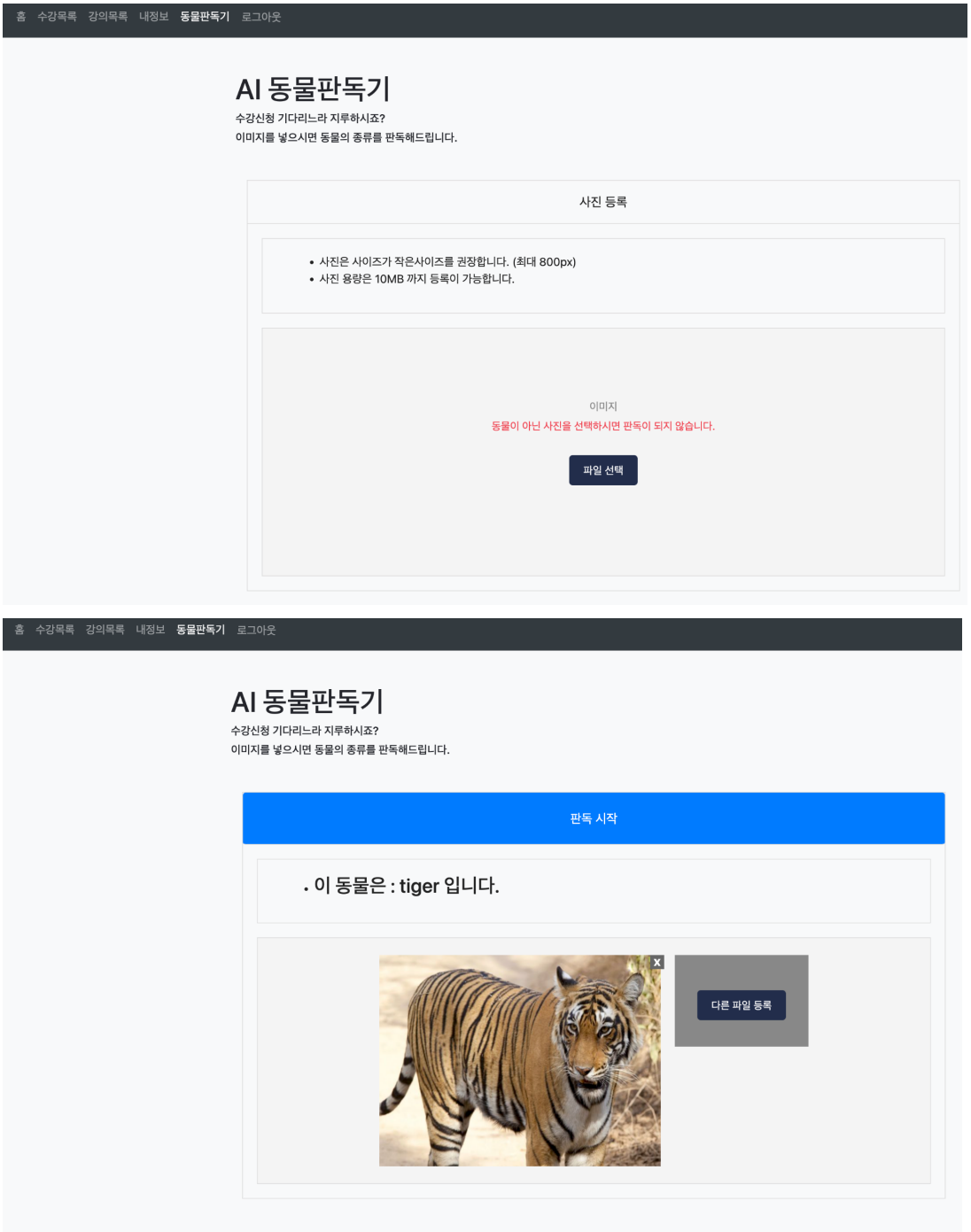


그림 26 User 동물판독기

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2. infrastructure

2.2.1. infrastructure 기획

2.2.1.1. infrastructure architecture 설계

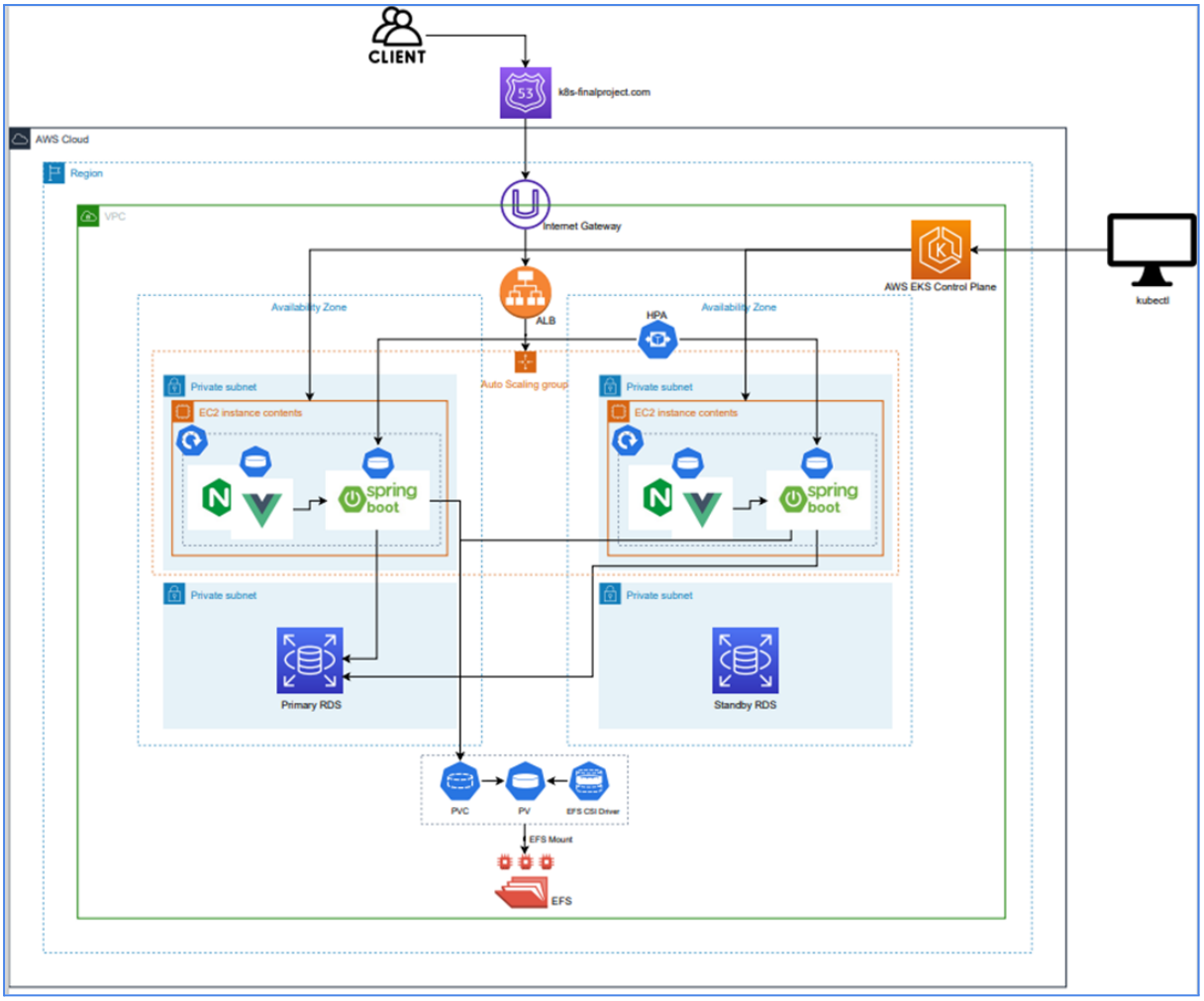


그림 27 infrastructure architecture

- 클러스터를 배포할 때 노드 그룹에 Capacity를 2로 설정하여 EC2 인스턴스가 각 AZ에 하나씩 생기게 하였다.

- 각 노드에는 백엔드와 프론트엔드 파드가 Deployment에 의해 배포된다.
- 고가용성을 위해 RDS 데이터베이스는 멀티 AZ 배포로 이중화 배치를 하였다.
- HPA를 사용하여 백엔드 서버의 파드에 연결한다. 이후 HPA는 파드의 request값으로 100%를 측정한다. 그리고 cpu 평균 사용률을 측정해서 지정한 값보다 높아지면 파드를 늘린다.
- 오토스케일링 그룹으로 두개의 EC2 인스턴스를 묶는다. 만약 파드가 HPA에 의해 계속 늘어나고 늘어난 파드의 request 값을 해당 노드가 가진 한정된 자원으로는 늘어난 파드의 request 값을 보장해주지 못한다면 오토스케일링 그룹에 의해 EC2 인스턴스(=노드)가 늘어나 새로운 노드에 이전에 배치하지 못하고 Pending 상태였던 파드가 배치된다.
- 백엔드 서버 파드에 PVC를 부착하여 스토리지 클래스에 의해 동적으로 PV를 생성해 NFS 스토리지에 연결시킨다. 이때 EFS를 사용하여 EFS Provisionor에 PV를 연결시켜서 EFS에 마운트시킨다. 따라서 여러 인스턴스가 동시적으로 스토리지에 접근해 읽기/쓰기가 가능해진다. 즉, 여러 인스턴스를 동기화 시킬 수 있다.
- 인스턴스와 RDS는 보안을 위해 각각 프라이빗 서브넷에 배치하고 Ingress(ALB)와 NodePort Service(NLB)를 이용하여 외부의 클라이언트에 노출시킨다.
- Route 53을 이용하여 외부 사용자가 도메인을 입력하여 서비스를 이용할 수 있게 한다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2.2. infrastructure 구축

2.2.2.1. EKS cluster

EKS 클러스터를 생성하는 방법은 콘솔창에서 생성, Cloud formation 또는 CDK 와 같은 IaC적 방법, EKS 공식 CLI인 eksctl로 배포하기와 같은 방법들이 있다. 이번 프로젝트에서는 eksctl을 사용하여 배포할 것이다.

Amazon EKS 클러스터를 생성하고 관리할 때 필요한 다음 도구 및 리소스를 설치하고 구성해야 한다.

- kubectl: Kubernetes 클러스터 작업을 위한 명령줄 도구이다.

- eksctl: 많은 개별 태스크를 자동화하는 EKS 클러스터를 사용하기 위한 명령줄 도구이다.

- 필요한 IAM 권한: 사용하는 IAM 보안 주체에는 Amazon EKS IAM 역할 및 서비스 연결 역할, AWS CloudFormation, VPC 및 관련 리소스를 사용할 수 있는 권한이 있어야 한다.

eksctl을 사용하여 변수 값 없이 클러스터 생성 명령어(eksctl create cluster)를 실행하면 기본값으로 클러스터가 배포된다. 따라서 구성 파일을 작성하여 배포할 것이다.

cluster

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: finalproject
  region: ap-northeast-2
  version: "1.24"

# AZ
availabilityZones: ["ap-northeast-2a", "ap-northeast-2b",  "ap-northeast-2c"]

# IAM OIDC & Service Account
iam:
  withOIDC: true
  serviceAccounts:
    - metadata:
        name: aws-load-balancer-controller
        namespace: kube-system
      wellKnownPolicies:
        awsLoadBalancerController: true
    - metadata:
        name: ebs-csi-controller-sa
        namespace: kube-system
      wellKnownPolicies:
        ebsCSIController: true
    - metadata:
        name: cluster-autoscaler
        namespace: kube-system
      wellKnownPolicies:
        autoScaler: true
    - metadata:
        name: efs-csi-controller-sa
        namespace: kube-system
      wellKnownPolicies:
        efsCSIController: true
```

그림 28 cluster.yaml (1)

```
# Managed Node Groups
managedNodeGroups:
  # On-Demand Instance
  - name: mynodes-t3
    instanceType: t3.medium
    minSize: 2
    desiredCapacity: 2
    maxSize: 5
    privateNetworking: true
    #ssh:
      #allow: true
      #publicKeyPath: ./keypair/myeks.pub
    availabilityZones: ["ap-northeast-2a", "ap-northeast-2b", "ap-northeast-2c"]
    iam:
      withAddonPolicies:
        autoScaler: true
        albIngress: true
        cloudWatch: true
        ebs: true

# Fargate Profiles
fargateProfiles:
  - name: myfg
    selectors:
    - namespace: dev
      labels:
        env: dev

# CloudWatch Logging
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
```

그림 29 cluster.yaml (2)

```
❯ kubectl get nodes
NAME                                               STATUS   ROLES    AGE   VERSION
ip-192-168-118-156.ap-northeast-2.compute.internal   Ready    <none>   28h   v1.24.10-eks-48e63af
ip-192-168-138-230.ap-northeast-2.compute.internal   Ready    <none>   28h   v1.24.10-eks-48e63af
ip-192-168-156-137.ap-northeast-2.compute.internal   Ready    <none>   14d   v1.24.10-eks-48e63af
ip-192-168-177-222.ap-northeast-2.compute.internal   Ready    <none>   13d   v1.24.10-eks-48e63af
```

그림 30 cluster.yaml (3)

- 인프라 아키텍처대로 두 개의 워커 노드(EC2 인스턴스)를 생성한다

- 각 EC2 인스턴스는 2a, 2b, 2c 가용 영역에 분포된다.

- Ingress와 HPA, 오토스케일링 그룹, 스토리지를 위한 OIDC 공급자를 통해 IAM 정책을 만들고 IAM 정책을 통해서 IAM 역할(서비스 계정)을 생성한다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2.2.2. Service deploy (문서의 흐름이 어떻게 진행되는지에 대한 흐름 서술 및 설명 필요)

📒 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

2.2.2.2.1. Frontend (Vue-user) deploy

vi front-deployment-user

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front-deploy-user
spec:
  replicas: 2
  selector:
    matchLabels:
      tier: frontend-user
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 0
  minReadySeconds: 20
  template:
    metadata:
      name: front-deploy-user
      labels:
        tier: frontend-user
    spec:
      containers:
      - name: front-app-user
        image: suhwan11/frontend-user:12
        ports:
        - containerPort: 80
          protocol: TCP
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: tier
                operator: In
                values:
                - frontend-user
            topologyKey: "kubernetes.io/hostname"
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: tier
                operator: In
                values:
                - backend
            topologyKey: "kubernetes.io/hostname"
```

그림 31 front-deployment-user

- 파드는 2개를 생성하도록 하였다. 파드 인스턴스를 점진적으로 새로운 것으로 업데이트하여 디플로이먼트 업데이트가 서비스 중단 없이 이루어질 수 있도록 해주는 롤링 업데이트 전략을 사용하였다.
    - 컨테이너 이미지로 Vue 빌드 파일을 nginx에 올려서 생성한 이미지를 도커 허브에서 가져오고 포트는 엔진엑스포트인 80으로 지정하였다.
    - 파드의 템플릿과 디플로이먼트 사이의 관계를 맺기위해 tier: frontend-user 레이블을 사용하였다.
- frontend 파드에 마찬가지로 PodAntiAffinity 설정으로 label이 tier: frontend인 파드를 배척해서 스케줄링(=배치) 되게 하였다. frontend파드가 아무리 늘어나도 파드들의 레이블은 동일하므로 frontend파드끼리 같은 노드에 배치되는 일은 존재하지 않는다. PodAffinity 설정으로 label이 tier: backend인 파드에 대해서 해당 파드와 묶여서 동일한 노드에 배치되도록 하였다. 즉, 서비스를 구성하는 backend파드와 frontend 파드가 한쌍이 되어서 각 노드(= EC2 인스턴스)에 배치된다. 단, 동일한 기능을 하는 파드가 묶여서 배치되는 일은 없다.

vi front-svc-user

```
apiVersion: v1
kind: Service
metadata:
  name: front-svc-user
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31112
  selector:
    tier: frontend-user
```

그림 32 front-svc-user

- NodePort는 외부에서 노드 IP의 특정 포트(<NodeIP>:<NodePort>)로 들어오는 요청을 감지하여, 해당 포트와 연결된 파드로 트래픽을 전달하는 유형의 서비스이다. 이때 클러스터 내부로 들어온 트래픽을 특정 파드로 연결하기 위한 ClusterIP 역시 자동으로 생성된다.

- 외부 트래픽을 내부에 전달하기 위해 서비스가 생성할 포트는 80, 서비스가 접근할 파드의 포트는 위에서 생성했던 컨테이너 포트인 80으로 맞춰준다.

- spec.selector에 해당하는 모든 파드들에 동일한 로드 밸런싱이 적용된다.

2.2.2.2.2. Frontend (Vue-Admin) deploy

vi front-admin-deploy

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front-deploy-admin
spec:
  replicas: 2
  selector:
    matchLabels:
      tier: frontend-admin
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 0
  minReadySeconds: 20
  template:
    metadata:
      name: front-deploy-admin
      labels:
        tier: frontend-admin
    spec:
      containers:
      - name: front-app-admin
        image: suhwan11/frontend-admin:7
        ports:
        - containerPort: 80
          protocol: TCP
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: tier
                operator: In
                values:
                - frontend-admin
            topologyKey: "kubernetes.io/hostname"
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: tier
                operator: In
                values:
                - backend
            topologyKey: "kubernetes.io/hostname"
```

그림 33 front-deployment-admin

- 파드의 템플릿과 디플로이먼트 사이의 관계를 맺기 위해 tier: frontend-admin 레이블을 사용하였다.

- 설정 자체는 위와 동일하다. 컨테이너 이미지로 admin용 Vue 빌드 파일을 nginx에 올려서 생성한 이미지를 도커 허브에서 가져오고, 포트는 nginx 포트인 80으로 지정하였다.

vi front-admin-svc

```
apiVersion: v1
kind: Service
metadata:
  name: front-svc-admin
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31113
  selector:
    tier: frontend-admin
```

그림 34 front-admin-svc

- 설정 자체는 위와 동일하다. 외부 트래픽을 내부에 전달하기 위해 서비스가 생성할 포트는 80, 서비스가 접근할 파드의 포트는 위에서 생성했던 컨테이너 포트인 80으로 맞춰준다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2.2.2.3. Frontend Ingress

vi front-ing

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: front-ing
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: instance
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:ap-northeast-2:506294220241:certificate/569
707b3-25d9-4c5b-8556-2db67c7b5013 , arn:aws:acm:ap-northeast-2:506294220241:certificate/650d118b-1704-
45b3-ac85-c1ee1f22b2fe
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}, {"HTTPS":443}]'
    alb.ingress.kubernetes.io/tls-redirect: '443'
spec:
  rules:
  - host: "k8s-finalproject.com"
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: front-svc-user
            port:
              number: 80
  - host: "k8s-finalproject-admin.com"
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: front-svc-admin
            port:
              number: 80
```

그림 35 front-ing

- Ingress를 사용하기 위해서는 LoadBalancer Controller 애드온을 설치해야 한다. 처음에 클러스터를 만들 때 서비스 계정과 IAM 애드온을 추가하였다.

- admin 페이지와 user 페이지에 따라 도메인 네임을 다르게 설정하였기 때문에 각각 호스트를 등록해서 정의해 주었다. 또한 prefix를 /로 설정하였기 때문에 도메인 네임만 입력했을 때 해당 어플리케이션으로 이동한다. 외부에서 서비스에 접근할 때 http 포트인 80 포트로 접근하기 때문에 80포트를 열어주었다.

- 어노테이션 부분에서 클러스터에 Kubernetes 수신 리소스가 생성될 때마다 ALB 및 필요한 지원 AWS 리소스를 생성한다. 또한 클러스터 내의 노드를 ALB의 대상으로 등록한다. ALB에 도달하는 트래픽은 서비스를 위해 pods로 라우팅된 다음 NodePort로 프록시된다.

- 어노테이션 부분에서 식별한 TLS 인증서의 arn을 각각 추가하고, listen-port로 80/http, 443/https 포트를 추가함으로써 외부사용자의 80/http OR 443/https 포트를 통한 요청을 둘다 허용한다. 이때 tls-redirect를 443으로 지정함으로써 80/http 요청에 대해 443/https로 리다이렉션한다.

2.2.2.2.4. Backend (Java Spring boot) deploy

back-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: back-deploy
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 0
  minReadySeconds: 20
  selector:
    matchLabels:
      tier: backend
  replicas: 2
  template:
    metadata:
      labels:
        tier: backend
    spec:
      volumes:
      - name: efs-volume
        persistentVolumeClaim:
          claimName: efs-pvc
      containers:
      - name: back-app
        image: suhwan11/backend:6
        ports:
        - containerPort: 8080
          protocol: TCP
        volumeMounts:
        - name: efs-volume
          mountPath: /data
        resources:
          requests:
            cpu: 200m
            memory: 200M
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: tier
                operator: In
                values:
                - backend
            topologyKey: "kubernetes.io/hostname"
```

그림 36 back-deployment

- 파드는 2개 생성하도록 하였다. 파드 인스턴스를 점진적으로 새로운 것으로 업데이트하여 디플로이먼트 업데이트가 서비스 중단 없이 이루어질 수 있도록 해주는 롤링 업데이트 전략을 사용하였다.

- 컨테이너 이미지로 스프링부트 애플리케이션을 gradle로 빌드하여 생성한 실행파일을 Java로 실행시킨 이미지를 도커 허브에서 가져오고 포트는 스프링 부트 포트인 8080으로 지정하였다.

- 파드의 템플릿과 디플로이먼트 사이의 관계를 맺기위해 tier: backtend 레이블을 사용하였다.

- PVC를 이용하여 볼륨을 생성하고 생성한 볼륨을 /data에 마운트하였다.

- backend 파드에 PodAntiAffinity 설정으로 label이 tier: backend인 파드를 서로 배척하도록 스케줄링 하였다. Backend 파드가 아무리 늘어나도 파드들의 레이블은 모두 동일하기 때문에 backend 파드끼리 같은 노드에 배치되는 일은 존재하지 않는다.

vi back-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: back-svc
spec:
  selector:
    tier: backend
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 31111
  type: NodePort
```

그림 37 back-svc

- NodePort는 외부에서 노드 IP의 특정 포트(<NodeIP>:<NodePort>)로 들어오는 요청을 감지하여, 해당 포트와 연결된 파드로 트래픽을 전달하는 유형의 서비스이다. 이때 클러스터 내부로 들어온 트래픽을 특정 파드로 연결하기 위한 ClusterIP 역시 자동으로 생성된다.

- 외부 트래픽을 내부에 전달하기 위해 서비스가 생성할 포트는 80, 서비스가 접근할 파드의 포트는 위에서 생성했던 컨테이너 포트인 8080으로 맞춰준다.

- spec.selector에 해당하는 모든 파드들에 동일한 로드 밸런싱이 적용된다.

back-ing

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: back-ing
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: instance
spec:
  rules:
  - http:
      paths:
      - pathType: Prefix
        path: /
        backend:
          service:
            name: back-svc
            port:
              number: 80
```

그림 38 back-ing

- Ingress를 사용하기 위해서는 LoadBalancer Controller 애드온을 설치해야 한다. 처음에 클러스터를 만들 때 서비스 계정과 iam 애드온을 추가했었다.

- prefix를 /로 설정하였기 때문에 도메인 네임만 입력했을 때 해당 어플리케이션으로 이동한다. 외부에서 서비스에 접근할 때 http 포트인 80포트로 접근하기 때문에 80포트를 열어주었다.

- 어노테이션 부분에서 클러스터에 Kubernetes 수신 리소스가 생성될 때마다 ALB 및 필요한 지원 AWS 리소스를 생성한다. 또한 클러스터 내의 노드를 ALB의 대상으로 등록한다. ALB에 도달하는 트래픽은 서비스를 위해 pods로 라우팅된 다음 NodePort로 프록시된다.

2.2.2.2.5. Backend (Python Flask) deploy

ml-deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-deploy
spec:
  replicas: 2
  selector:
    matchLabels:
      tier: ml
  template:
    metadata:
      name: ml-deploy
      labels:
        tier: ml
    spec:
      containers:
      - name: ml-deploy
        image: suhwan11/test1:flaskv2
        ports:
        - containerPort: 5000
```

그림 39 ml-deployment

- 컨테이너 이미지로 flask 빌드 파일을 python으로 실행하여 생성한 이미지를 도커 허브에서 가져오고 포트는 flask 포트인 5000으로 지정하였다.

- 파드의 템플릿과 디플로이먼트 사이의 관계를 맺기위해 tier: ml 레이블을 사용하였다.

ml-svc

```
apiVersion: v1
kind: Service
metadata:
  name: ml-svc
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 5000
    nodePort: 31117
  selector:
    tier: ml
```

그림 40 ml-svc

- NodePort는 외부에서 노드 IP의 특정 포트(<NodeIP>:<NodePort>)로 들어오는 요청을 감지하여, 해당 포트와 연결된 파드로 트래픽을 전달하는 유형의 서비스이다. 이때

클러스터 내부로 들어온 트래픽을 특정 파드로 연결하기 위한 ClusterIP 역시 자동으로 생성된다.

- 외부 트래픽을 내부에 전달하기 위해 서비스가 생성할 포트는 80, 서비스가 접근할 파드의 포트는 위에서 생성했던 컨테이너 포트인 5000으로 맞춰준다.

- spec.selector에 해당하는 모든 파드들에 동일한 로드 밸런싱이 적용된다.

ml-ing

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ml-ing
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: instance
spec:
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: ml-svc
            port:
              number: 80
```

그림 41 ml-ing

- prefix를 /로 설정하였기 때문에 도메인 네임만 입력했을 때 해당 어플리케이션으로 이동한다. 외부에서 서비스에 접근할 때 http 포트인 80포트로 접근하기 때문에 80포트를 열어주었다.
- 어노테이션 부분에서 클러스터에 Kubernetes 수신 리소스가 생성될 때마다 ALB 및 필요한 지원 AWS 리소스를 생성한다. 또한 클러스터 내의 노드를 ALB의 대상으로 등록한다. ALB에 도달하는 트래픽은 서비스를 위해 pods로 라우팅된 다음 NodePort로 프록시된다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2.2.3. EFS Storage

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 고 가용성을 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 있으므로 공통 데이터 소스를 액세스 및 공유할 수 있기 때문에 EFS를 선택하였다.

추후 스토리지 클래스가 지정된 PVC를 파드에 마운트해서 PV를 동적으로 프로비저닝해서 PV가 EFS에 요청할 수 있게 만들 것이다.

이를 위해서는 EFS에 요청할 수 있는 PV가 필요하고 이러한 PV의 프로파일 정보를 가진 스토리지 클래스가 필요하다.

따라서 EFS CSI 드라이버를 설치하고 설치한 드라이버를 이용하여 스토리지 클래스를 생성하면 추후에 이 스토리지클래스를 이용하여 PVC를 생성하고 파드에 마운트하면, 파드는 PVC에 의해 동적으로 생성된 PV가 EFS에 요청을 통해 스토리지와 상호작용을 할 수 있게 된다.

여기에서는 위에서 생성한 서비스 계정을 이용하여 EFS-CSI 드라이버를 설치하는 과정을 이미 진행했다고 가정하고 넘어가겠다. 해당 내용은 AWS 홈페이지에서 확인할 수 있다.

EFS 스토리지 클래스 생성

efs-storage-class

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: efs-sc
parameters:
  directoryPerms: "700"
  fileSystemId: fs-07b34570f35c4cc5d
  provisioningMode: efs-ap
provisioner: efs.csi.aws.com
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

그림 42 efs-storage-class

- 맨 처음 클러스터를 배포할 때 EFS 서비스 계정을 생성해주었다.

- fileSystemId에 위에서 생성한 EFS의 ID를 추가해준다.

- 프로비저너로 EFS-CSI 드라이버를 선택한다.

PVC 생성

efs-pvc

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 1Gi
```

그림 43 efs-pvc

- 위에서 생성한 스토리지 클래스를 지정한다.

2.2.2.4. HPA

HPA(Horizontal Pod Autoscaler)는 Kubernetes에서 사용되는 자동 스케일링 기능 중 하나이다. HPA를 사용하면, 클러스터 내 파드 수를 자동으로 조정하여, 요구 사항에 따라 애플리케이션의 성능을 최적화할 수 있고, 파드 수를 증가시켜, 더 많은 트래픽을 처리하거나, 응답 시간을 줄이는 등의 이점이 있기에, HPA를 사용하기로 하였다.
HPA는 CPU, Memory 등 리소스가 정해둔 임계치를 초과할 경우 자동으로 스케일 아웃 해주는 기능을 갖추고 있다. HPA 컨트롤러가 리소스를 체크하며 정해둔 replicas 수에 맞춰 파드를 줄이거나 늘려준다.

HPA를 사용하기 위해서는 Metrics server 애드온을 설치해야 하는데, 이는 클러스터를 배포할 때 설정해 두었다. 또한 HPA를 사용하기 위해 deployment에 request 값을 설정해줘야 한다.

back-hpa.yaml

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: back-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: back-deploy
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

그림 44 back-hpa2.2.2.5 Cluster Autoscaler

기존의 EKS클러스터는 Backend 서버를 위한 파드 두개를 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.

min 값은 2 , max 값은 5 , desired 값은 2로 지정하였다. 즉, 현재 배포된 클러스터의 노드의 개수는 2개이고, 2개이하로는 scale down되지 않게 설정하였고, 최대 늘어날 수 있는 노드는 5개로 설정하였다. **(*min**값과 **max**값을 **2**와 **5**로 설정한 이유에 대한 설명 필요**)**

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

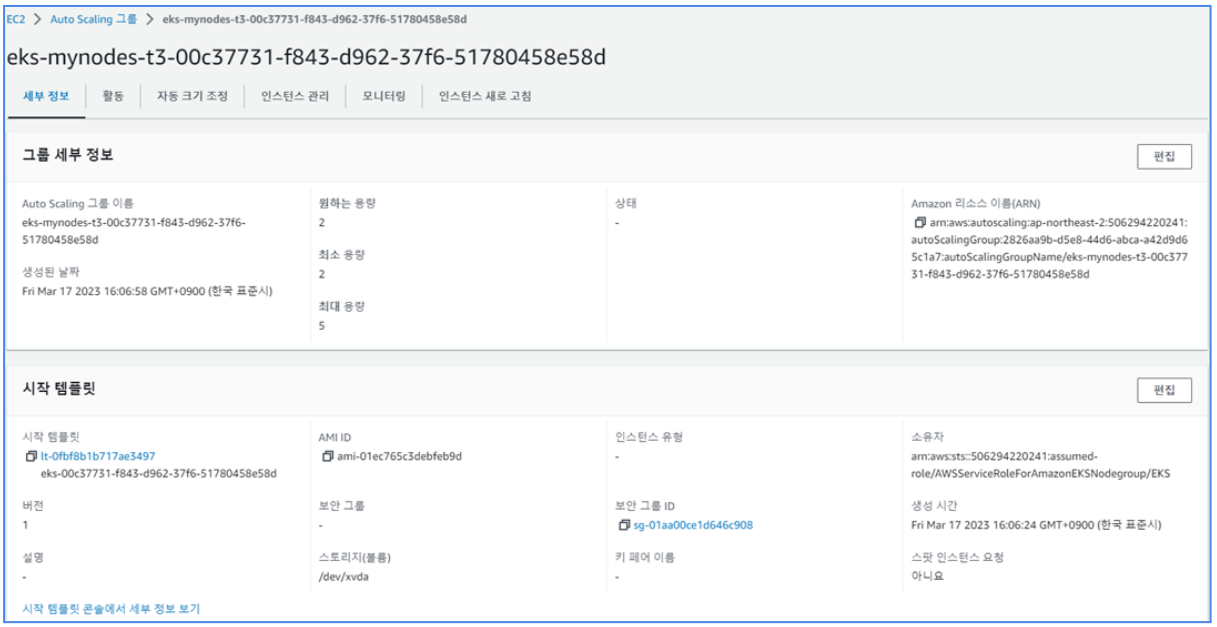


그림 45 Cluster Autoscaler

2.2.2.5. ArgoCD

ArgoCD 설치는 다음 홈페이지로 가면 설치 방법에 관해 알 수 있다.
(https://argo-cd.readthedocs.io/en/stable/)

ArgoCD 설치를 위한 명령어는 다음과 같다.

```
# 네임 스페이스 생성
kubectl create namespace argocd

# argo CD 생성
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

# 로드밸런서 외부용 IP 확인
kubectl get svc -n argocd

# 로드밸런서로 바꿈
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'

# 초기 패스워드 찾기
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d; echo
```

그림 46 ArgoCD command

처음 ArgoCD 콘솔에 접속하면 로그인을 해야 한다. 초기 계정은 admin이고 패스워드는 위의 명령으로 확인한 패스워드로 로그인 하면 된다. 로그인 후 User Info에서 패스워드 변경 가능하다. 이후 ArgoCD에서 새로운 앱을 생성하여 서비스용 yaml 파일들이 모여있는 git 저장소와 연동해주면 다음과 같이 잘 배포되는 것을 확인할 수 있다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

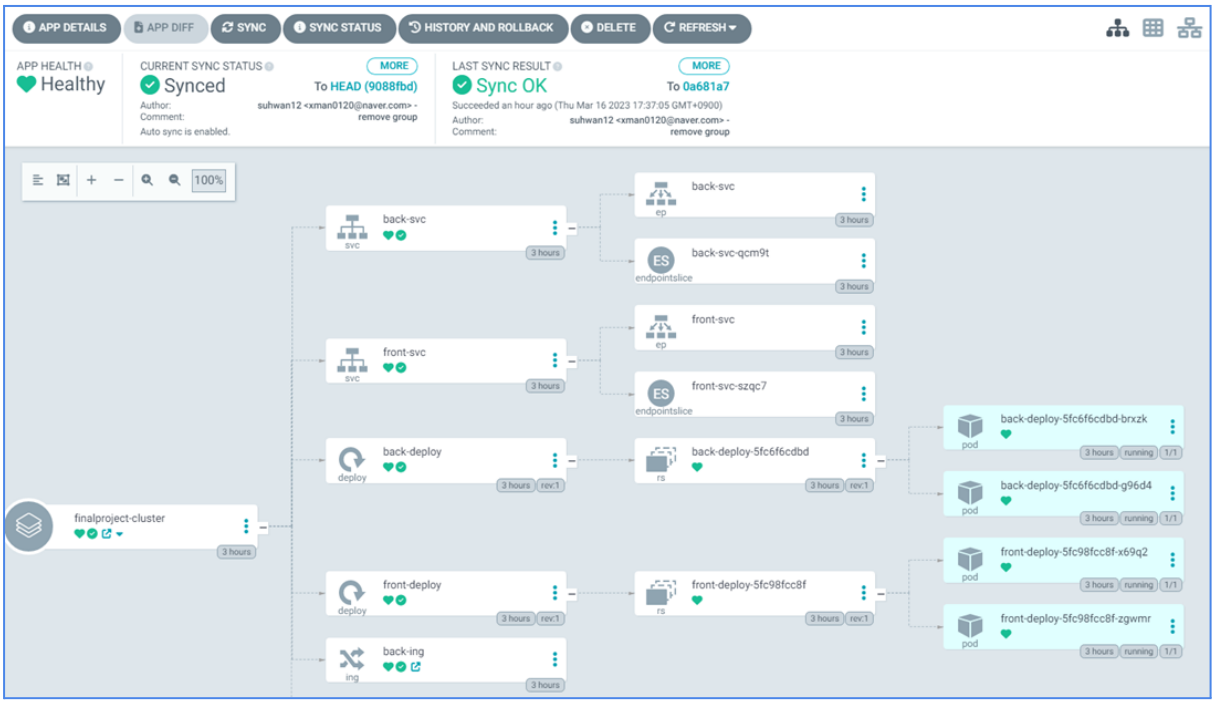


그림 47 ArgoCD Dashborad

2.2.2.6. EKS-RDS 연동 및 생성

RDS 인스턴스와 EKS 클러스터를 연동하는 목적은 서버 인스턴스와 DB 인스턴스를 분리하여 사용하기 위한 목적으로, AWS를 사용하는 핵심 이유이기도 하다.

연동과정은 이런 방식으로 진행된다.

1) 연동할 EKS 정보 파악
2) DB 보안 그룹 생성
3) RDS용 서브넷 생성
4) RDS 서브넷 그룹 생성
5) 파라미터 그룹 생성
6) DB생성
7) 라우팅 테이블 작성

1) 연동할 **EKS** 정보 파악

우리가 사용해야할 EKS, 보안 그룹, 서브넷 그룹, RDS 인스턴스를 모두 같은 네트워크 안에서 사용되야 하므로 EKS에서 사용하고 있는 VPC와 서브넷 ID를 파악한다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

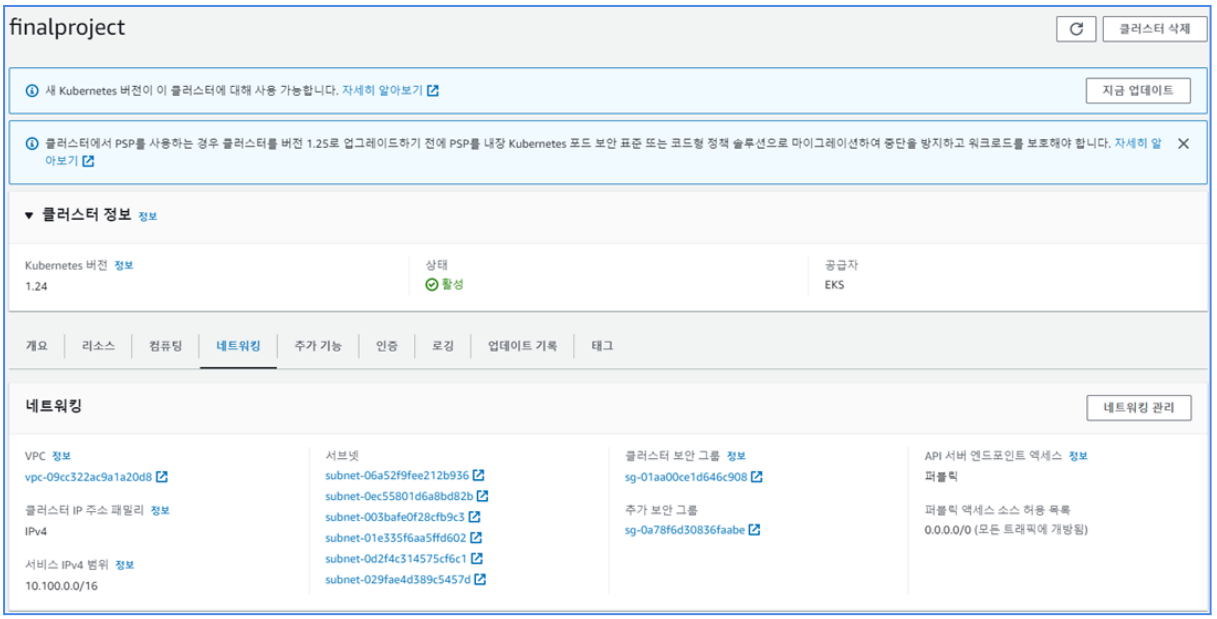


그림 48 EKS 정보

가용 영역별로 public, private 서브넷이 각각 생성되어 있다.

또한 인스턴스에 들어가 클러스터로 사용하고 있는 인스턴스에서 어떤 보안 그룹을 사용하고 있는지 확인한다. 2개의 노드가 모두 같은 보안 그룹을 사용하고 있는 것을 확인해야 한다.

2) DB 보안 그룹 생성

RDS DB에서 사용할 DB 보안 그룹을 생성한다. [EC2]에서 왼쪽 메뉴바의 [네트워크 및 보안]의 [보안 그룹]을 선택하고, 보안 그룹 생성을 클릭한다. 보안 그룹의 이름과 설명을 각각 적당하게 써준 뒤, VPC는 EKS와 동일한 VPC 대역으로 설정해준다.

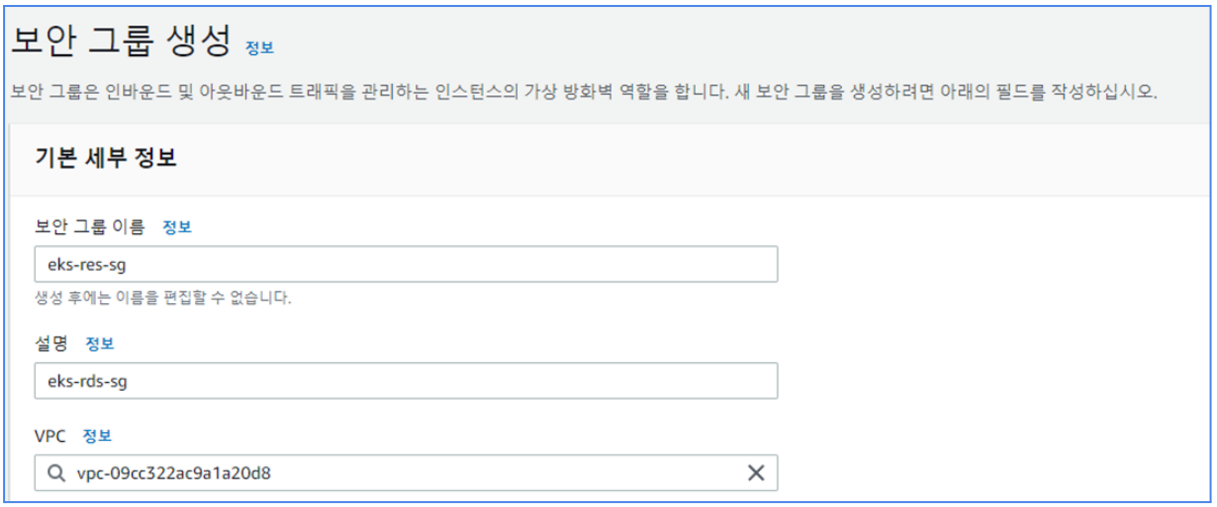


그림 49 보안그룹 생성

로컬 PC에서 RDS로 접근하기 위해서는 RDS의 보안 그룹에 본인 PC의 IP를 인바운드에 추가해야 한다.

인바운드 규칙에서 MySQL을 사용할 것이기에, MySQL을 선택하면 포트는 3306으로 지정된다.

인바운드에 추가할 항목은 다음과 같다.

- 내 PC의 IP

- EC2에 사용된 보안 그룹의 그룹 ID

EKS에 사용된 보안 그룹을 RDS의 보안 그룹 규칙에 추가하는 이유는 이렇게 해야 EKS에서 RDS로 접속도 가능하기 때문이다. 또한 개인 로컬 PC에서 RDS 인스턴스에 직접 접속할 수 있도록 내 공인 IP도 등록해준다.

3) RDS용 서브넷 생성

기존에 존재하는 서브넷 6개는 생성된 VPC의 IP인 192.168.0.0/16 을 서브넷 마스크를 /19로 서브넷팅하여 대역을 분리한 것이다. 따라서 현재 사용할 수 있는 서브넷의 범위가 2^3 - 6 인 2개가 남아있기 때문에 이를 이용해서 RDS용 private 서브넷 2개를 생성할 것이다. 가용 영역은 어디에든 설정해도 괜찮다. 대신 2개의 서브넷의 가용 영역만 다르게 설정하면 된다.

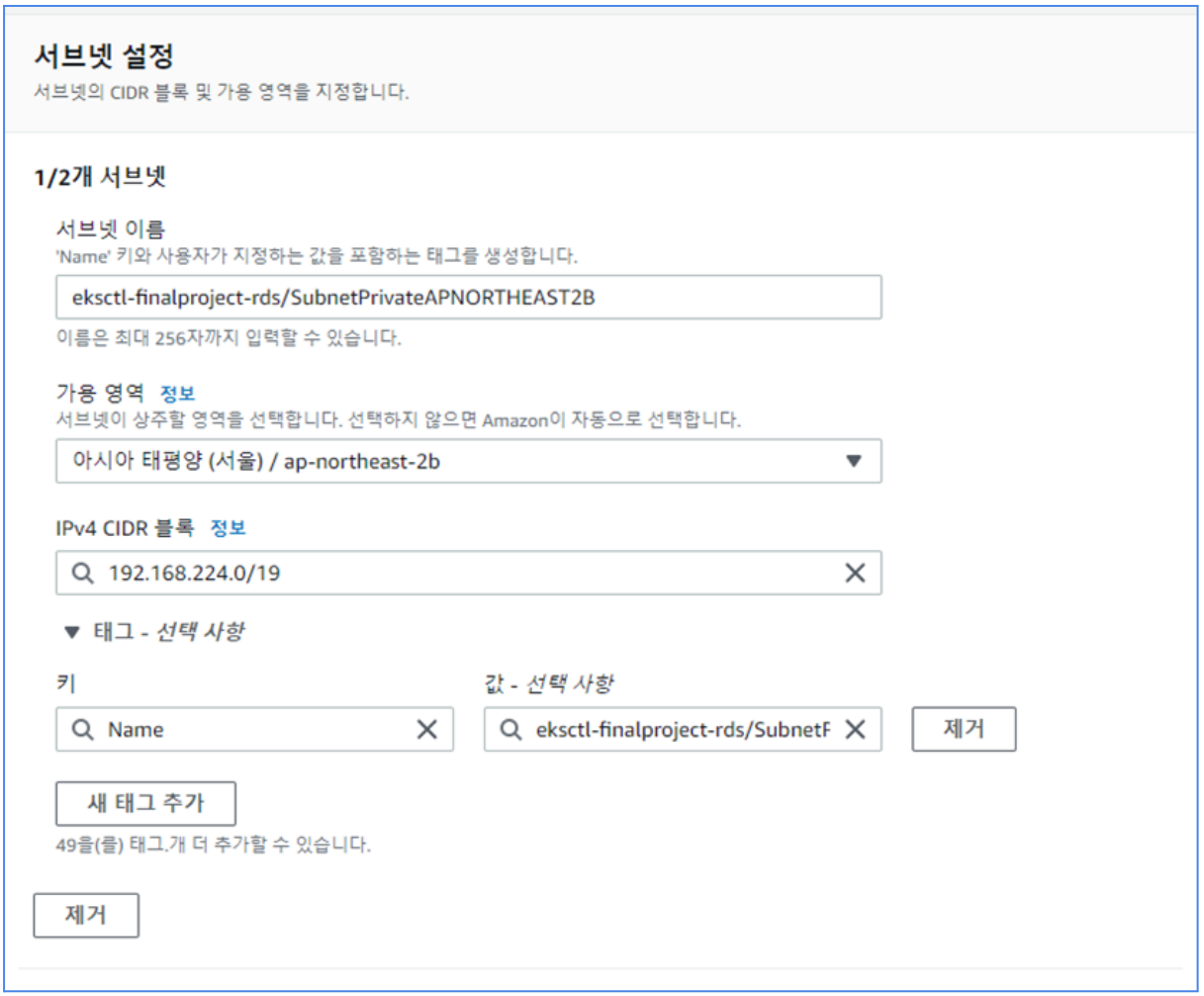


그림 50 RDS용 서브넷 생성 (1)

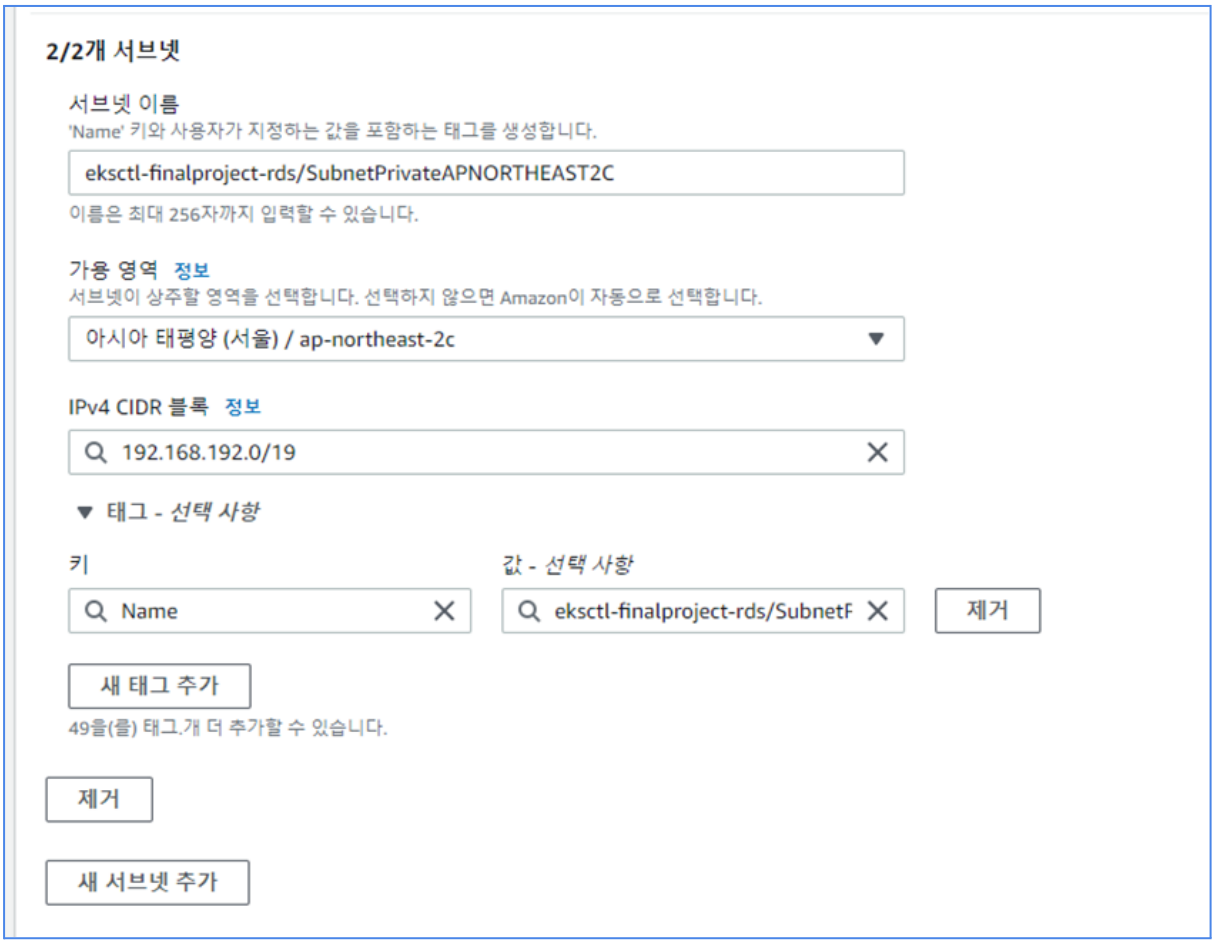


그림 51 RDS용 서브넷 생성 (2)

4) RDS 서브넷 그룹 생성

[RDS]에서 [서브넷 그룹] 메뉴를 선택하고 DB 서브넷 그룹 생성 버튼을 클릭한다. 서브넷 그룹 이름을 생성하고 VPC를 이번에도 마찬가지로 EKS에서 사용하고 있는 VPC로 선택한다. 서브넷을 추가할 때 정했던 가용 영역을 선택한 후 만들었던 서브넷을 선택하고 만들어 주면 된다.

이때 EKS Cluster 내부에 Private Subnet에 구축하기 위해 가용 영역과 하단 서브넷은 Private Subnet만 선택한다.

그림 52 RDS 서브넷 그룹 생성

5) 파라미터 그룹 생성

파라미터 그룹을 생성하는 이유는 한글 사용을 가능하게 하기 위함과 DB의 시간을 한국 시간으로 설정하기 위함이다.

[RDS]에서 [파라미터 그룹] 메뉴를 선택하고 파라미터 그룹 생성 버튼을 클릭한다.
파라미터 이름을 설정하고 DB 패밀리도 지정해준 후 생성한다.

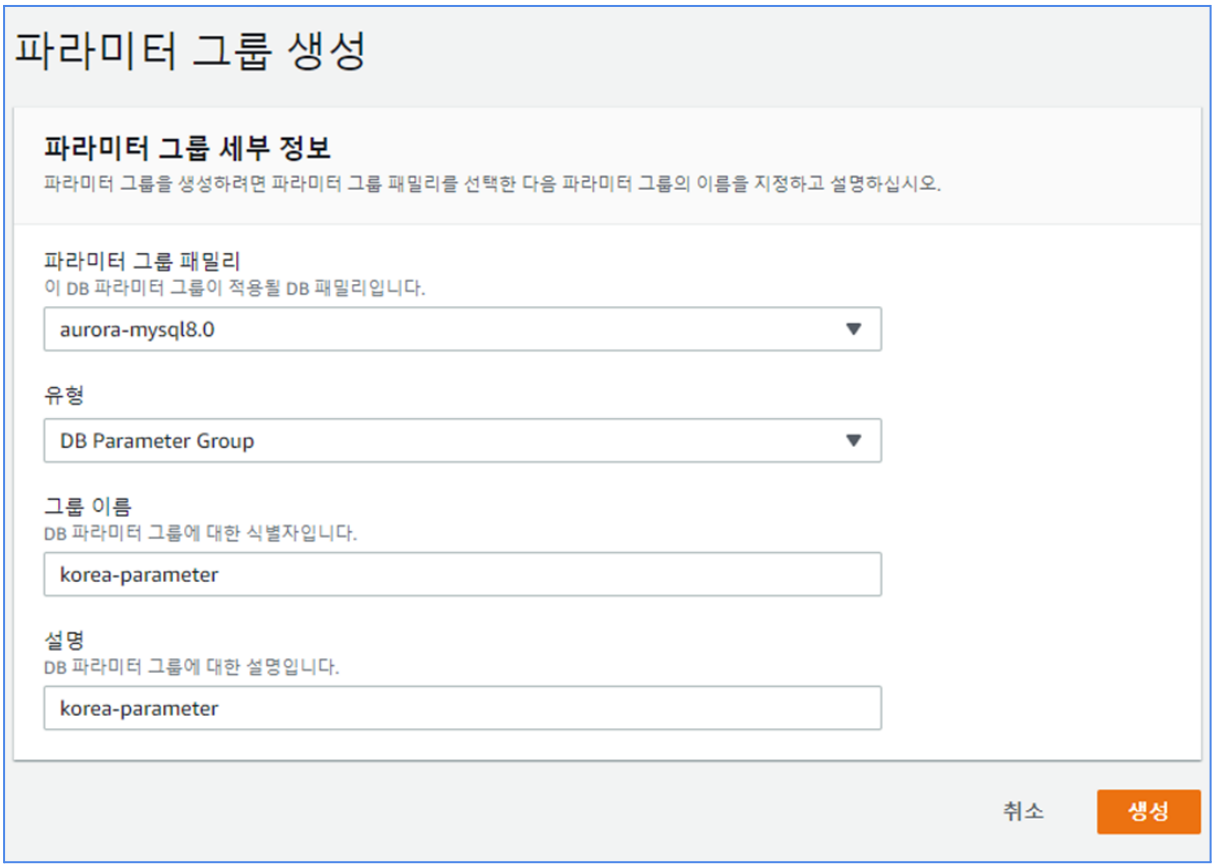


그림 53 파라미터 그룹 생성 (1)

파라미터 그룹에서 korean-parameter 가 생성됨을 확인할 수 있다. 이후 방금 생성된 파라미터 그룹을 누르고 파라미터 그룹 작업의 편집을 누른다. char을 검색하고 모든 값을 UTF-8으로 바꾼다.

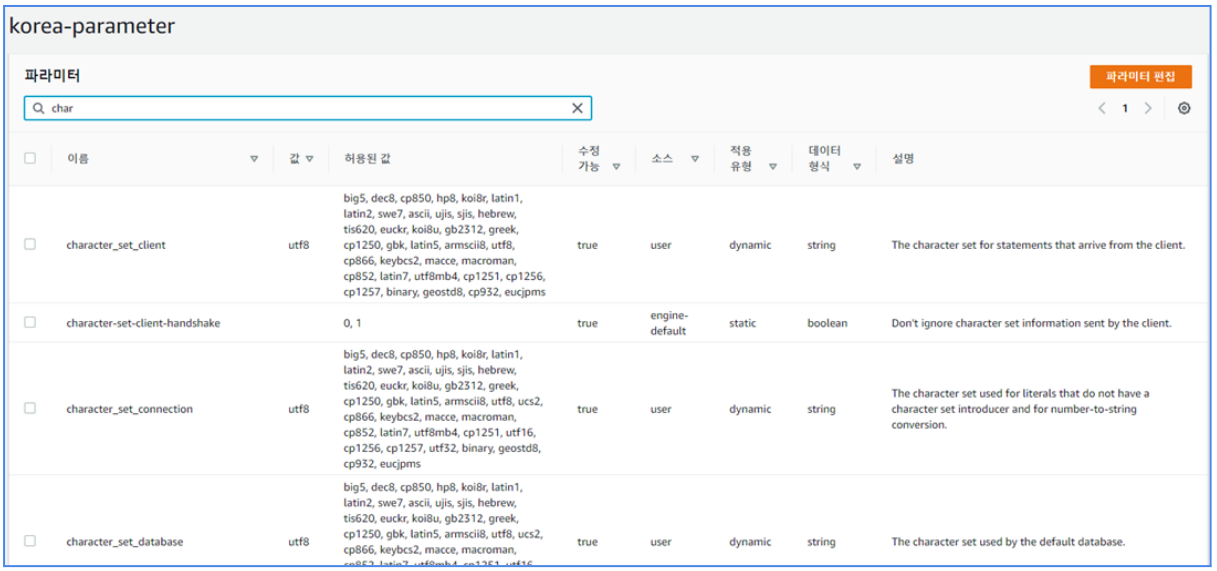


그림 54 파라미터 그룹 생성(2)

collation을 검색하고 이번에는 나와 있는 모든 값을 utf8_general_ci로 바꾼다. DB 시간 설정을 위해 이번에는 zone 이라고 검색하고, time_zone을 Asia/Seoul로 바꾼다. 모든 편집을 완료한 후 저장을 누르면 끝이다.

6) RDS 생성

[RDS]에서 [데이터베이스] 메뉴를 선택한다. 데이터베이스 생성 과정은 다음과 같다.

- MySQL를 사용할 것이므로 MySQL 선택

- 고가용성을 위해서 다중 AZ DB 인스턴스를 이용할 것이기에 개발/테스트 템플릿을 선택

- MySQL에 접속할 때 사용할 마스터의 이름과 비밀번호를 입력

- VPC : EKS Cluster VPC 선택, 서브넷 그룹 : 기존에 만들었던 그룹 선택, 퍼블릭 액세스 : '예', VPC 보안 그룹 : 기존에 만들었던 eks-rds-sg 선택

- 나머진 그대로, 추가구성에서 DB 파라미터 그룹만 만들었던 korea-parameter로 변경

이후 데이터베이스 생성을 진행해주면 된다.

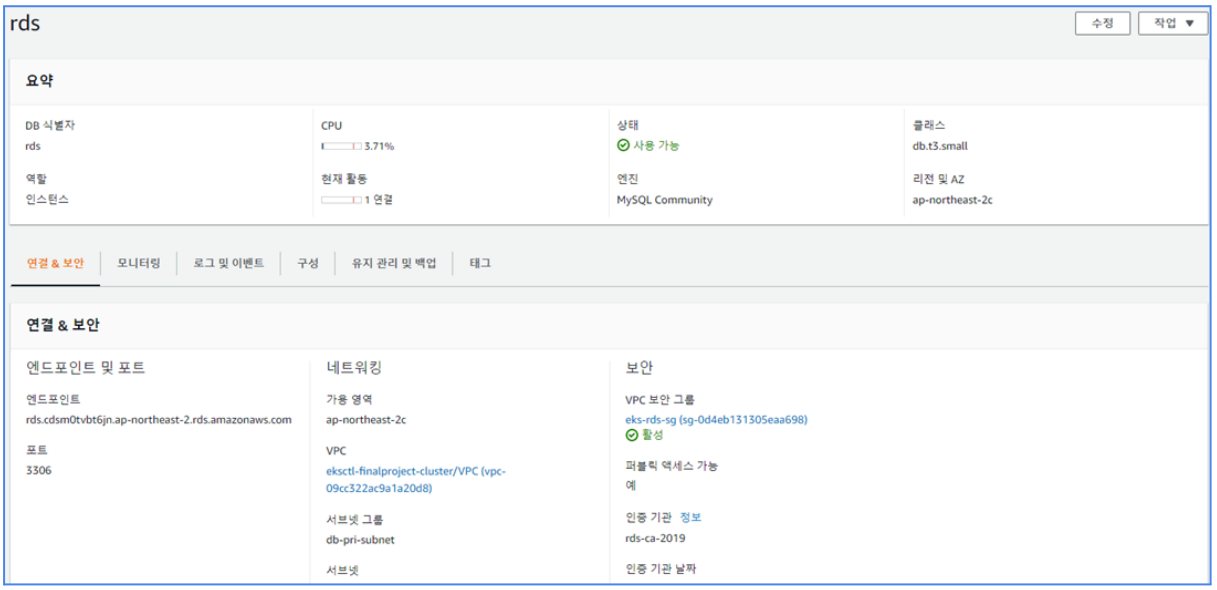


그림 55 RDS 생성

7) 라우팅 테이블 작성

새로 만든 서브넷들의 라우팅 테이블을 설정해줄 것이다. RDS용으로 만든 서브넷 2개에 대한 라우팅 테이블을 각각 만들어준다.

그 후 생성한 라우팅 테이블을 클릭한 후 라우팅에 들어간다. 기본적으로 vpc에서 로컬로 가는 라우팅 테이블이 존재할 것이다.

라우팅 편집으로 들어가 라우팅 경로를 몇 개 추가해준다. 추가할 항목은 프라이빗 서브넷이기 때문에 NAT 게이트웨이로 가는 것, 그리고 개발자와 내 공인 IP을 대상으로 인터넷 게이트웨이로의 연결을 추가하고 저장해주면 된다.

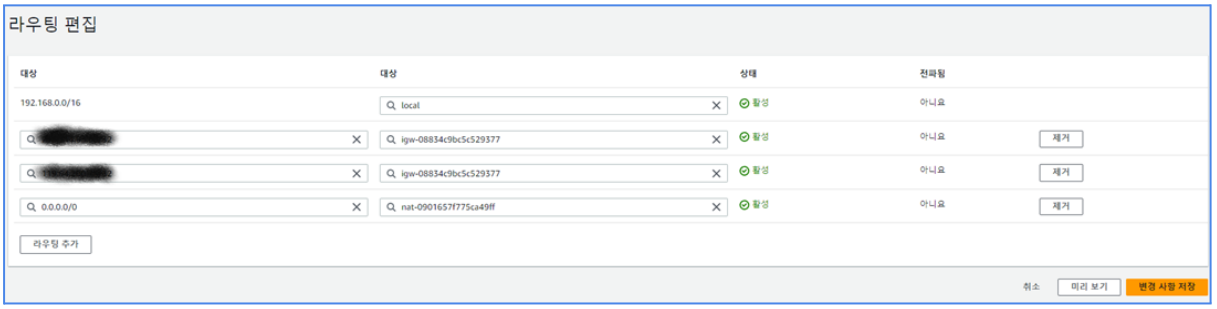


그림 56 라우팅 테이블 작성

같은 라우팅 테이블에서 서브넷 연결로 이동하면 명시적 연결이 없는 서브넷으로 생성한 프라이빗 서브넷이 존재할 것이다. 라우팅 설정을 마친 서브넷을 이름에 맞게 각각 명시적 서브넷 연결에 등록해주면 된다.

※ 내 로컬 PC(외부)에서 MySQL 접속

MySQL의 대표적인 클라이언트인 MySQL Workbench에서 접속을 시도해볼 것이다.

우선 AWS RDS 정보 페이지에서 엔드 포인트를 확인한다. 이 엔드 포인트가 접근 가능한 URL이므로 복사해둔다.

Hostname에 복사해두었던 RDS 엔드포인트를 입력하고 데이터베이스를 생성했을 때 설정했던 마스터의 이름과 비밀번호를 입력한다.

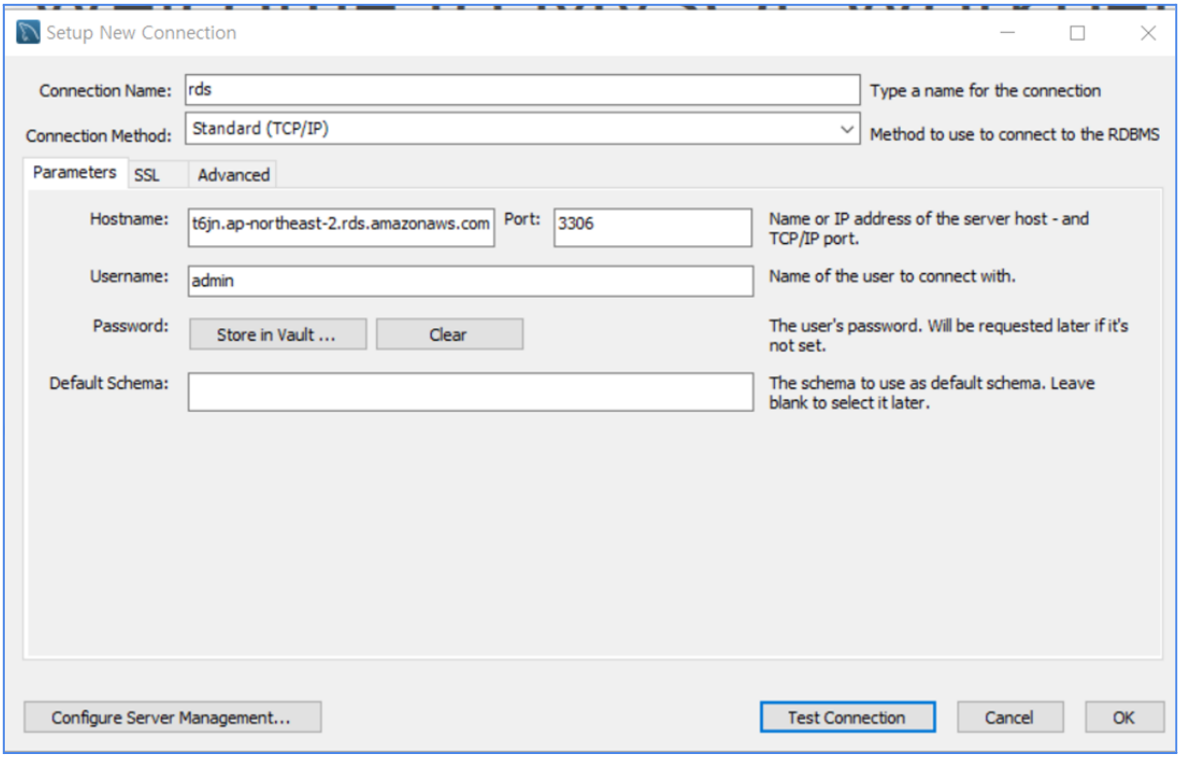


그림 57 외부에서 MySQL 접속 (1)

Test Connetion을 클릭하면 성공적으로 MySQL에 접속하는 것을 볼 수 있다.

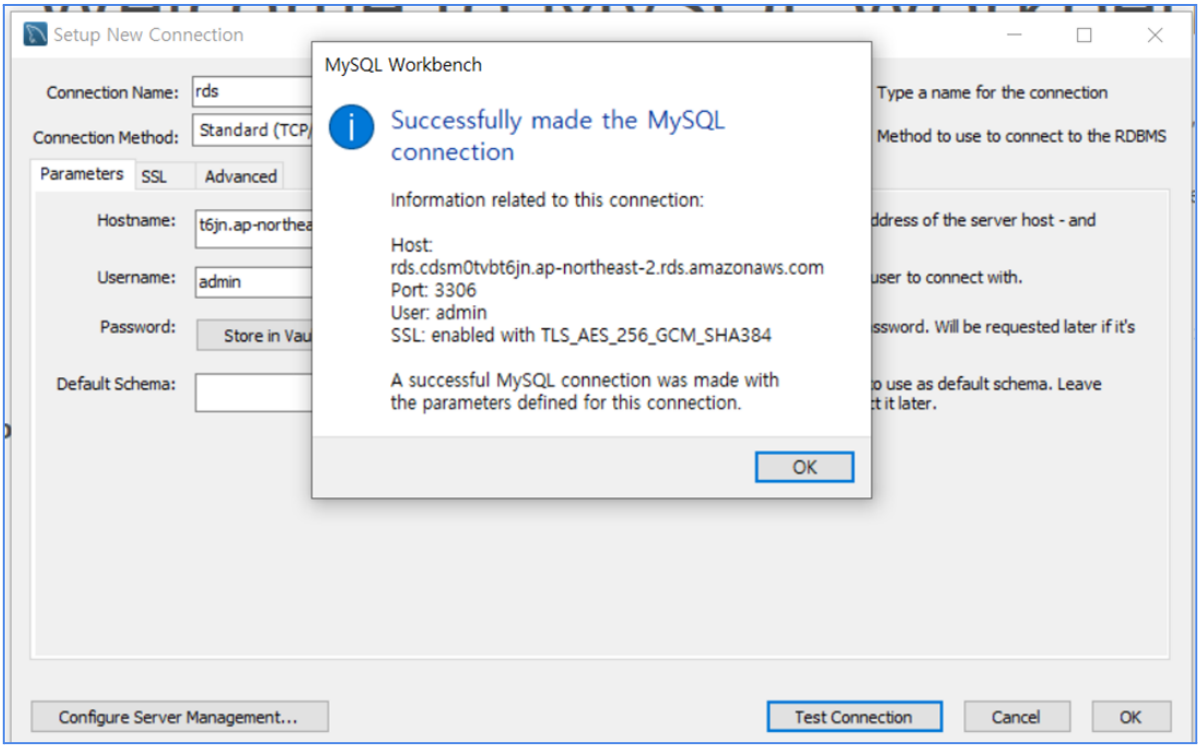


그림 58 외부에서 MySQL 접속 (2)

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2.3. CI/CD

2.2.3.1. pipeline architecture 설계

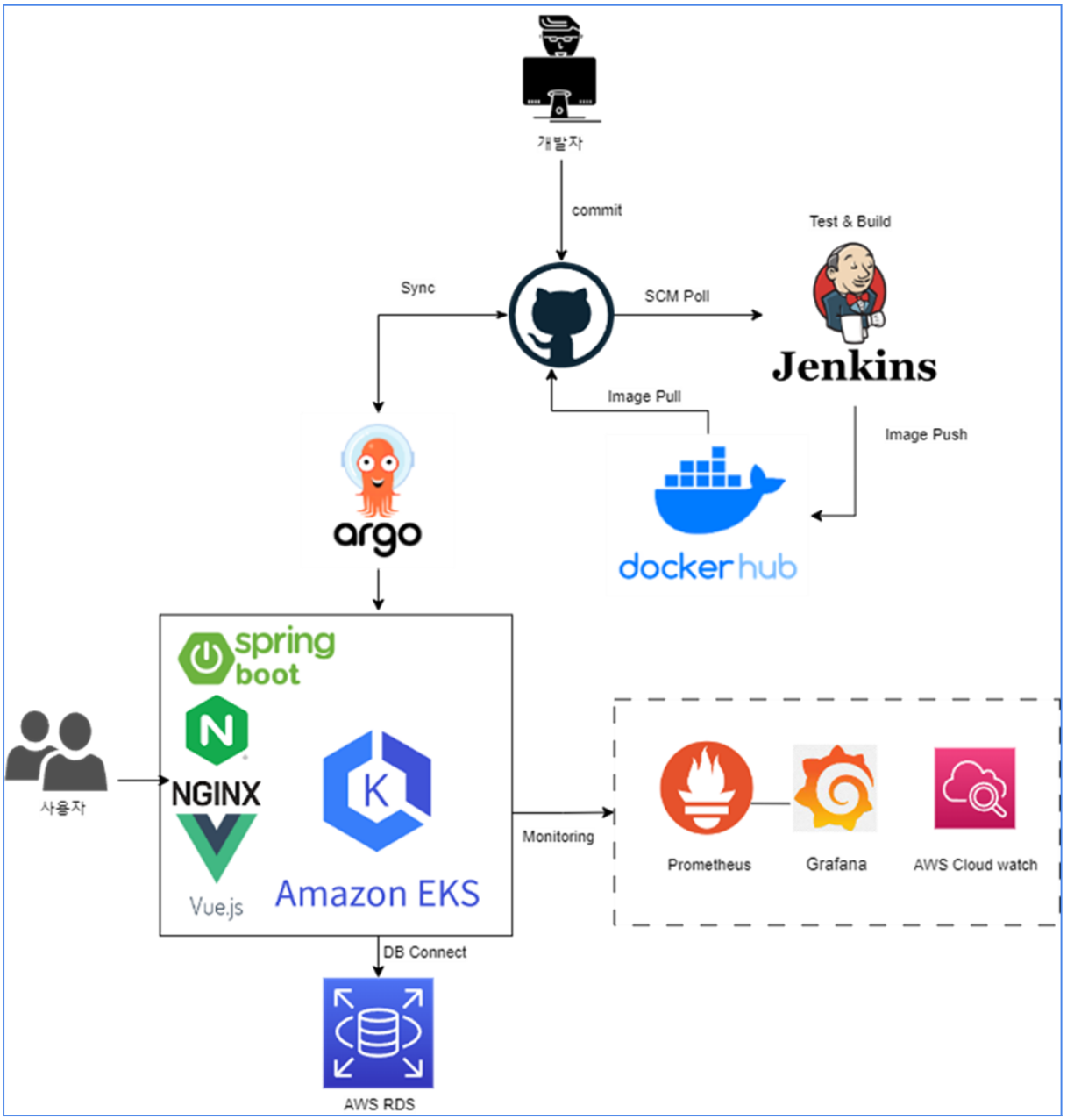


그림 59 Pipeline Architecture

- 개발자가 Jenkins에서 SCM Poll하고 있는 git에 개발한 코드를 Push하면 Jenkins는 SCM Poll을 통해 해당 코드를 가져와서 Test와 Build를 하게 된다.

- 백엔드의 경우 성공적으로 Build가 되었다면 결과물로 jar파일이 생성되게 된다. 이 jar 파일을 배포하고, Java기반의 Web Application을 실행시키기 위해 특정 디렉터리에 옮기고 실행시키는 작업을 진행하는 Dockerfile을 git에 추가한다.

- 프론트엔드의 경우 성공적으로 Build가 되었다면 빌드되어 생성된 폴더 전체와 nginx의 설정을 수정하는 파일을 nginx의 특정 디렉터리에 옮기고 실행시켜주는 Dockerfile을 git에 추가한다.

- 이후 Kaniko를 이용해 이 Dockerfile을 Build하여 이미지로 만든 후, 생성된 이미지를 Docker Hub에 Push한다.

- 그림에서는 하나의 git으로 되어있지만 실제로는 배포용 git 저장소와 코드용 git 저장소가 분리되어있다.

- Docker Hub는 배포용 Git에 연동되어 있고, 새로운 이미지가 Push가 되면 배포용 Git의 Deployment에서 기존의 이미지에 Sed 명령을 통해 새로운 Image로 바꾼다.

- 새로운 Image로 바뀐 Deployment를 ArgoCD가 감지하여 새로운 Image가 적용된, 즉 새로운 버전의 웹으로 Rolling Update를 통해 배포한다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2.3.2. Backend Pipeline 구축

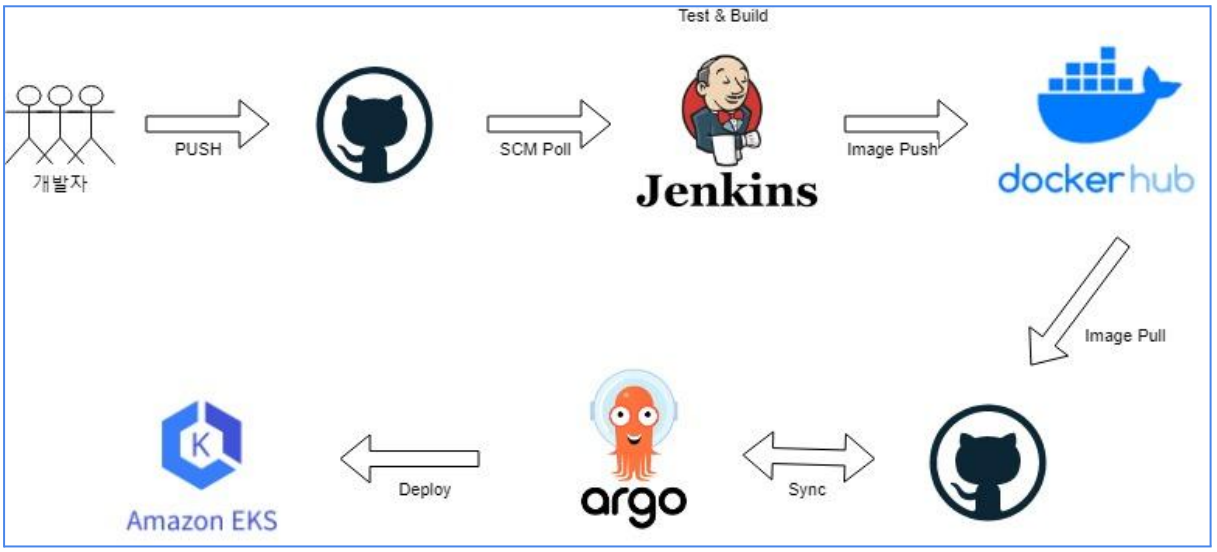


그림 60 pipeline

- 전체적인 Pipeline의 대략적인 플로우이다.
- Jenkins의 각 Pipeline은 개발자의 git에 연동되어있다.
- 각 git에는 Jenkinsfile이 존재한다.

2.2.3.2.1. Backend 도커라이징

- Jenkinsfile을 생성하기 앞서 EC2를 생성하고 접속하여 가상환경에서 Dockerifle을 생성한다.
- 이후 생성한 Dockerfile을 빌드하여 이미지가 정상적으로 생성되는지 테스트하고 생성한 이미지로 컨테이너를 배포하여 브라우저에 인스턴스의 퍼블릭IP를 입력했을 때 원하는 결과물이 나타나는지 확인한다.

2.2.3.2.1.1. Backend - RDS연동

```
spring:
  datasource:
    url:
jdbc:mysql://rds-pri.cdsm0tvbt6jn.ap-northeast-2.rds.amazonaws.com/test?
useSSL=false&useUnicode=true&serverTimezone=Asia/Seoul&allowPublicKeyRet
rieval=true
    username: admin
    password: qwer1234
    driver-class-name: com.mysql.cj.jdbc.Driver
```

- Backend 코드의 /src/main/resources/appilication-aws.yaml파일에 정의된 기존의 DB는 개발자의 로컬 DB다.
- 따라서 도커라이징 하기전에 위의 yaml파일에 RDS를 등록해줌으로써 Backend와 RDS를 연동시킨다.

2.2.3.2.1.2. Dockerfile

```
FROM gradle:8.0.2-jdk11
WORKDIR /usr/src/
RUN gradle wrapper --gradle-version 8.0.2
RUN ./gradlew build

FROM openjdk:11-jre
WORKDIR /usr/src/
COPY . /usr/src/
CMD ["java","-jar","/usr/src/build/libs/api-0.0.1-SNAPSHOT.jar"]
```

- Backend코드는 Spring Boot js를 이용하여 작성되었기 때문에 빌드하기 위해서는 gradle이미지를 가져온다.
- gradle을 이용하여 빌드할 수 있지만 어떤 개발환경이든 별도의 gradle설치없이도 빌드할 수 있는 gradlew를 설치한다.
- gradlew를 통해 Dockerfile을 빌드한다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

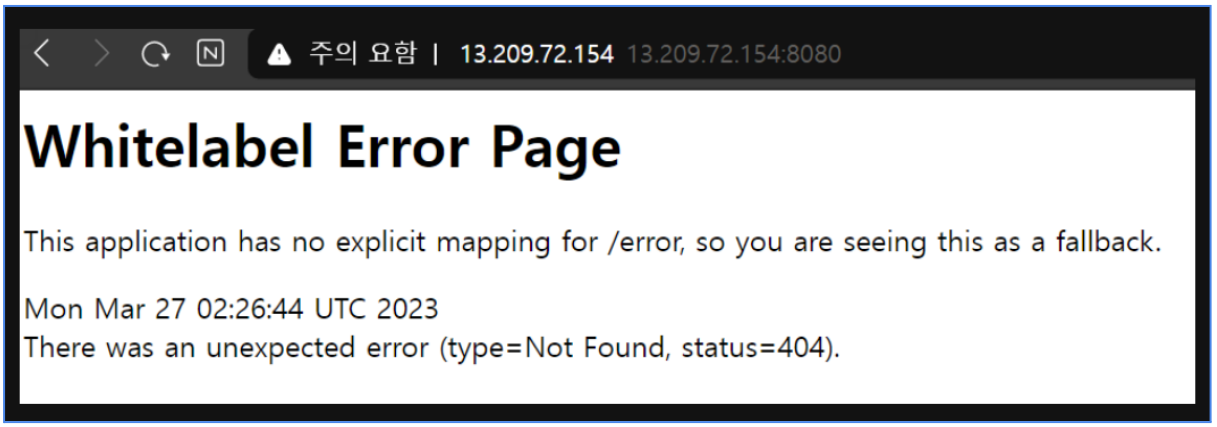


그림 61 Whitelabel Error Page

- 빌드를 통해 생성된 이미지로 컨테이너를 생성하여 배포하면 브라우저에 인스턴스 퍼블릭IP를 입력했을때 위와 같이 페이지가 나타나는 것을 확인
- Whitelabel Error Page란 Frontend가 아직 연결이 안되어 기본 설정 페이지가 나타나는 것이다.

2.2.3.2.2. Jenkins를 이용한 Pipeline

- Jenkins에 접속하여 Pipeline을 선택후 새로운 item을 생성한다.
- 새로운 item의 구성에 Backend코드를 가지고 있는 개발자의 git을 등록한다.
- 마찬가지로 앞으로 작성할 Jenkinsfile을 등록한다.

2.2.3.2.2.1. Jenkinsfile

```
pipeline{
    agent{
        kubernetes{
yaml '''
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: gradle
    image: gradle:8.0.2-jdk11
    command: ['sleep']
    args: ['infinity']
  - name: kaniko
```

```
    image: gcr.io/kaniko-project/executor:debug
    command: ['sleep']
    args: ['infinity']
    volumeMounts:
    - name: registry-credentials
      mountPath: /kaniko/.docker
  volumes:
  - name: registry-credentials
    secret:
      secretName: regcred
      items:
      - key: .dockerconfigjson
        path: config.json
'''
      }
    }

    stages{
        stage('checkout'){
          steps{
            container('gradle'){
              git branch: 'main', url:
'https://github.com/suhwan12/course-registration-GoormUniversity-restapi
.git'
              }
            }
      }
      stage('install gradlew'){
          steps{
            container('gradle'){
              sh 'gradle wrapper --gradle-version 8.0.2'
              }
          }
      }

        stage('gradle build project'){
            steps{
              container('gradle'){
                sh './gradlew build'
                }
```

```
            }
        }
        stage('Build & Tag docker image'){
            steps{
                container('kaniko'){
                    sh "executor --dockerfile=Dockerfile \
                    --context=dir://${env.WORKSPACE} \
                    --destination=suhwan11/backend:latest \
                    --destination=suhwan11/backend:${env.BUILD_NUMBER}"

                }
            }
        }
        stage('Update K8s to New Backend Deployment'){
            steps{
                container('gradle'){
                    git branch: 'main' ,
url:'https://github.com/suhwan12/finalproject-argocd.git'
                    sh 'sed -i "s/image:.*/image:
suhwan11\\/backend:${BUILD_NUMBER}/g" back-deployment.yaml'
                    sh 'git config --global user.name suhwan12'
                    sh 'git config --global user.email
xman0120@naver.com'
                    sh 'git config --global --add safe.directory
/home/jenkins/agent/workspace/Backend-pipeline'
                    sh 'git add back-deployment.yaml'
                    sh 'git commit -m "Jenkins Build Number -
${BUILD_NUMBER}"'
                    withCredentials([gitUsernamePassword(credentialsId:
'github-credentials', gitToolName: 'Default')]) {
                        sh 'git push origin main'
                    }
                }
            }
        }
    }
}
```

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

- kaniko로 빌드한 Dockerfile의 이미지를 Docker hub에 Push하기 위해서는 접근권한이 필요하다. 따라서 Docker Hub에서 Token을 발급받아 Credentials를 생성한다.
- 생성한 Credentials를 볼륨을 이용하여 Agent Pod의 Kaniko 컨테이너에 부착한다
- 마찬가지로 생성한 이미지를 Jenkins가 ArgoCD가 연동중인 git에 접근해 Sed명령어로 이미지를 교체하고 add , commit , push하기 위해서는 git에 대한 권한이 있어야한다.
- 따라서 git에서 token을 발급받고 jenkins - manage credentials에 들어가 해당 token을 이용하여 새로운 credentials를 생성한다.
- 생성한 credentials를 이용하여 마지막 stage에서 push를 한다.

2.2.3.2.2.2. Pipeline 결과 테스트

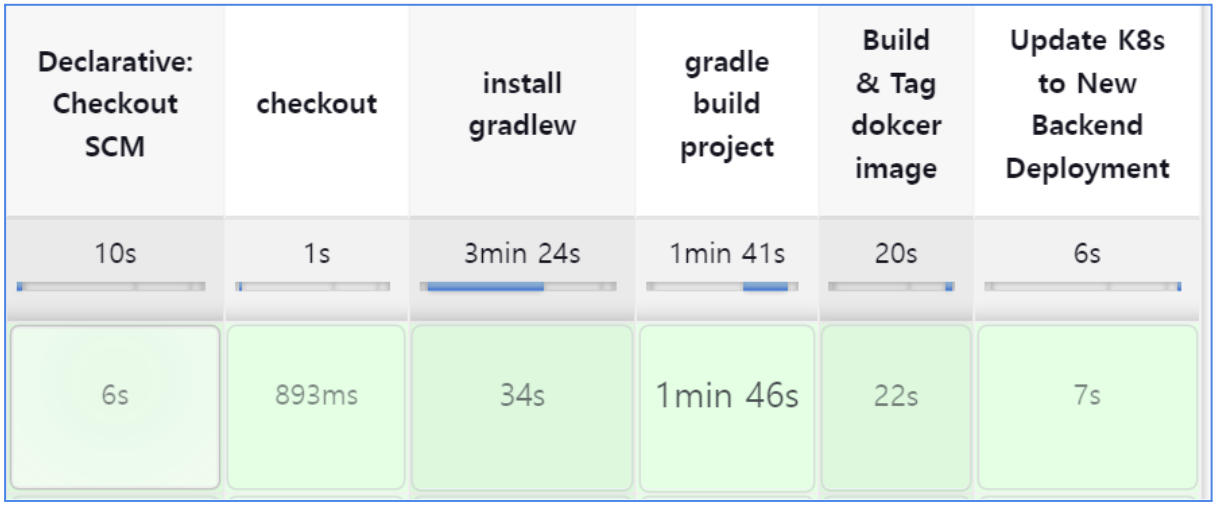


그림 62 *Pipeline* 결과 테스트

- Pipeline을 빌드했을때 정상적으로 빌드되는 것을 확인

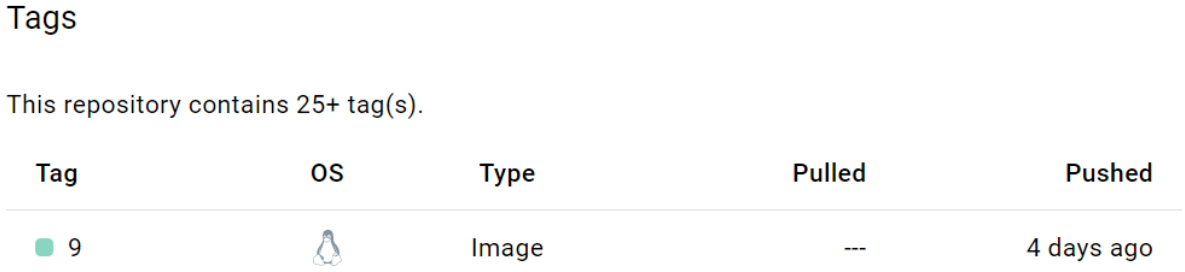
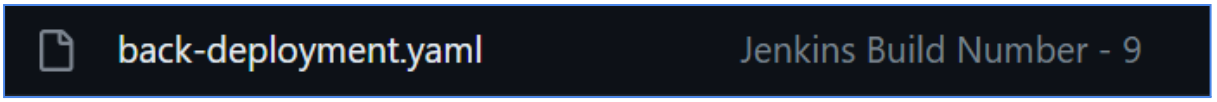


그림 63 Docker Hub Image

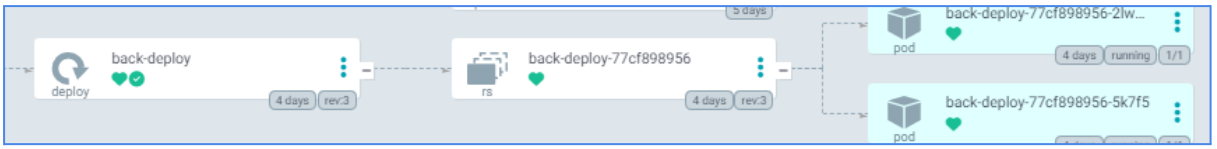
- Docker Hub에 image가 Push된 것을 확인



- ArgoCD가 연동중인 git에 backend파드를 생성하고있는 back-deployment.yaml파일에 새로운 commit이 생긴것을 확인

```
containers:
- name: back-app
  image: suhwan11/backend:9
```

- 해당 yaml파일에 들어가보면 실제로 image가 교체되어있는 것을 확인



- ArgoCD가 git에 변경사항이 생긴것을 감지하고 변경사항에 맞게 새롭게 배포한다.
- 새롭게 배포할때 새로운 replicaset을 만들고 그 아래 새로운 이미지가 적용된 컨테이너를 가진 파드를 Rolling Update를 통해 배포한다.

2.2.3.3. Frontend - User Pipeline 구축

- 위의 Backend Pipeline의 플로우와 동일하다.
- 마찬가지로 인스턴스에 접속하여 도커라이징 테스트 후 정상적으로 이미지 생성 및
- 컨테이너 배포가 완료되면 Pipeline을 구축할 것이다.

2.2.3.3.1. nginx에서 Frontend - Backend 연동

```
2.2.3.3.1. nginx에서 Frontend - Backend 연동
location /api {
    proxy_pass
http://k8s-default-backing-8476de3e44-889811917.ap-northeast-2.elb.amazo
naws.com;
    proxy_buffer_size   128k;
    proxy_buffers       4 256k;
    proxy_busy_buffers_size 256k;
  }
  location /mlapp {
    proxy_server:{
      proxy:{
        "/api":{
          target:
"http://k8s-default-backing-8476de3e44-889811917.ap-northeast-2.elb.amaz
onaws.com/api/",
          rewrite: (path)=>path.replace(/^\/api/,""),
        },
        "/mlapp":{
          target:
"http://k8s-default-mling-8ad2ca83ed-1581733671.ap-northeast-2.elb.amazo
naws.com/mlapp/",
          rewrite: (path)=>path.replace(/^\/mlapp/,""),
        },
      }
    }
```

- Frontend 와 Backend를 연동하기 위해서는 nginx와 Frontend설정 파일 모두에 Backend주소를 등록해주어야 한다.
- nginx.conf 파일을 생성해서 경로를 구분하여 기본 Backend 주소와 Machine Learning Backend 주소를 등록해준다.

2.2.3.3.2. Frontend 설정파일에서 Frontend - Backend 연동

```
server:{
      proxy:{
```

```
        "/api":{
          target:
"http://k8s-default-backing-8476de3e44-889811917.ap-northeast-2.elb.amaz
onaws.com/api/",
          rewrite: (path)=>path.replace(/^\/api/,""),
        },
        "/mlapp":{
          target:
"http://k8s-default-mling-8ad2ca83ed-1581733671.ap-northeast-2.elb.amazo
naws.com/mlapp/",
          rewrite: (path)=>path.replace(/^\/mlapp/,""),
        },
      }
    }
```

- 마찬가지로 vite.config.js 설정파일에서도 경로를 구분하여 기본 Backend 주소와 Machine Learning Backend 주소를 등록해준다.

2.2.3.3.3. Frontend - User 도커라이징

```
pipeline{
    agent{
        kubernetes{
yaml '''
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: gradle
    image: gradle:8.0.2-jdk11
    command: ['sleep']
    args: ['infinity']
  - name: kaniko
    image: gcr.io/kaniko-project/executor:debug
    command: ['sleep']
    args: ['infinity']
    volumeMounts:
    - name: registry-credentials
```

```
      mountPath: /kaniko/.docker
  volumes:
  - name: registry-credentials
    secret:
      secretName: regcred
      items:
      - key: .dockerconfigjson
        path: config.json
'''
      }
    }

    stages{
        stage('checkout'){
          steps{
            container('gradle'){
              git branch: 'main', url:
'https://github.com/suhwan12/course-registration-GoormUniversity-user.gi
t'
              }
            }
        }
        stage('npm install & npm Build & Docker Build & Tag Docker
image'){
          steps{
            container('kaniko'){
                    sh "executor --dockerfile=Dockerfile \
                    --context=dir://${env.WORKSPACE} \
                    --destination=suhwan11/frontend-user:latest \

--destination=suhwan11/frontend-user:${env.BUILD_NUMBER}"
                }
            }
        }
        stage('Update K8s to New Frontend Deployment'){
          steps{
            container('gradle'){
                git branch: 'main' ,
url:'https://github.com/suhwan12/finalproject-argocd.git'
                sh 'sed -i "s/image:.*/image:
```

```
suhwan11\\/frontend-user:${BUILD_NUMBER}/g" front-deployment-user.yaml'
                sh 'git config --global user.name suhwan12'
                sh 'git config --global user.email xman0120@naver.com'
                sh 'git config --global --add safe.directory
/home/jenkins/agent/workspace/Frontend-Pipeline-User'
                sh 'git add front-deployment-user.yaml'
                sh 'git commit -m "Jenkins Build Number -
${BUILD_NUMBER}"'
                withCredentials([gitUsernamePassword(credentialsId:
'github-credentials', gitToolName: 'Default')]) {
                    sh 'git push origin main'
                }
            }
          }
        }
    }
}
```

- 코드 수정을 통해 Backend - Frontend 연동이 끝났다면 위와같이 Dockerfile 작성한다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

그림 64 학생 로그인

- Dockerfile을 빌드 후 이미지가 생성되고 컨테이너로 배포한 후 브라우저에 인스턴스 퍼블릭 IP를 입력했을 때 정상적으로 User 페이지가 나타나는 것을 확인한다.

2.2.3.3.4. Jenkins를 이용한 Pipeline

```
pipeline{
    agent{
        kubernetes{
yaml '''
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: gradle
    image: gradle:8.0.2-jdk11
    command: ['sleep']
    args: ['infinity']
```

```
  - name: kaniko
    image: gcr.io/kaniko-project/executor:debug
    command: ['sleep']
    args: ['infinity']
    volumeMounts:
    - name: registry-credentials
      mountPath: /kaniko/.docker
  volumes:
  - name: registry-credentials
    secret:
      secretName: regcred
      items:
      - key: .dockerconfigjson
        path: config.json
'''
      }
    }

    stages{
        stage('checkout'){
          steps{
            container('gradle'){
              git branch: 'main', url:
'https://github.com/suhwan12/course-registration-GoormUniversity-user.gi
t'
              }
            }
        }
        stage('npm install & npm Build & Docker Build & Tag Docker
image'){
          steps{
            container('kaniko'){
                    sh "executor --dockerfile=Dockerfile \
                    --context=dir://${env.WORKSPACE} \
                    --destination=suhwan11/frontend-user:latest \

--destination=suhwan11/frontend-user:${env.BUILD_NUMBER}"
                }
            }
        }
```

```
        stage('Update K8s to New Frontend Deployment'){
          steps{
            container('gradle'){
                git branch: 'main' ,
url:'https://github.com/suhwan12/finalproject-argocd.git'
                sh 'sed -i "s/image:.*/image:
suhwan11\\/frontend-user:${BUILD_NUMBER}/g" front-deployment-user.yaml'
                sh 'git config --global user.name suhwan12'
                sh 'git config --global user.email xman0120@naver.com'
                sh 'git config --global --add safe.directory
/home/jenkins/agent/workspace/Frontend-Pipeline-User'
                sh 'git add front-deployment-user.yaml'
                sh 'git commit -m "Jenkins Build Number -
${BUILD_NUMBER}"'
                withCredentials([gitUsernamePassword(credentialsId:
'github-credentials', gitToolName: 'Default')]) {
                    sh 'git push origin main'
                }
            }
          }
        }
    }
}
```

- Backend Pipeline과 마찬가지로 새로운 item 생성 및 Frontend - User 코드가 있는 git을 연동한다.
- 위와 같이 Jenkinsfile을 작성한다.
- Jenkinsfile에서 git checkout stage와 마지막 stage에서 컨테이너로 gradle을 사용하였는데, Frontend - User 코드는 vue.js로 작성되었기 때문에 사실상 nodejs 이미지로 생성된 컨테이너 환경에서 작업하는 것이 맞다. 하지만 install과 빌드과정을 각 stage에서 작업하는 것이 아닌, Dockerfile에서 Multi Builder를 이용하여 한꺼번에 작업을 하기 때문에 굳이 다른 stage에서 nodejs 컨테이너를 사용할 필요가 없다.
- gradle 컨테이너를 이용하여 작업할 수 있는 노드, 즉 환경만 만들어주고 checkout과 마지막 stage를 해결한다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2.3.3.4.1. Pipeline 결과 테스트

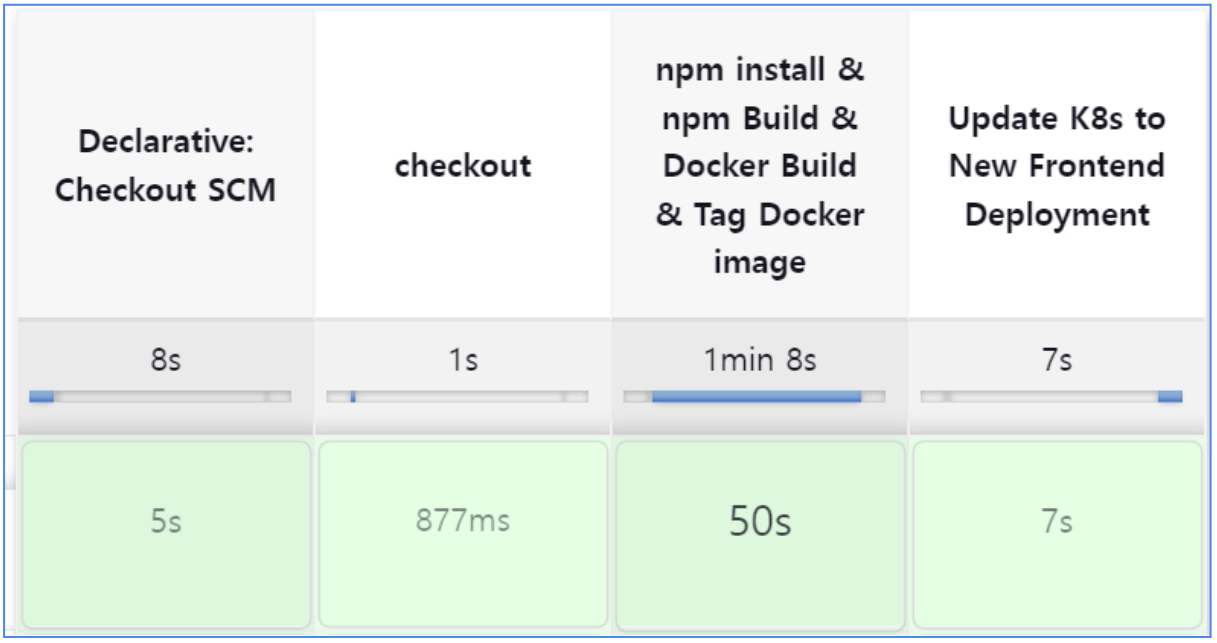


그림 65 *Pipeline* 결과 테스트

- Frontend - User Pipeline을 빌드하였을 때 정상적으로 빌드되는 것을 확인

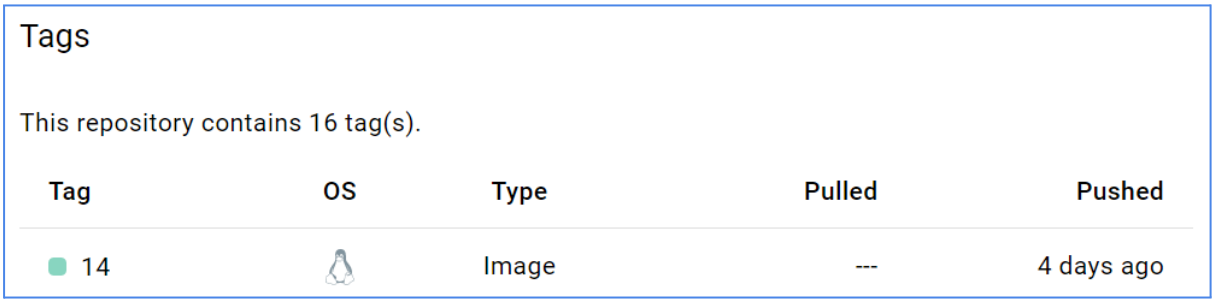


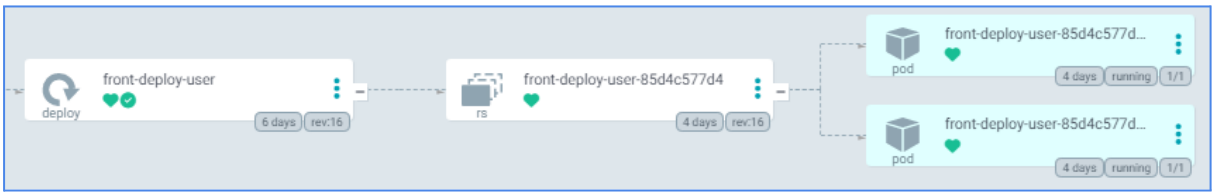
그림 66 Frontend - User Pipeline

- Docker Hub에 이미지가 Push된 것을 확인

- ArgoCD가 연동중인 git에 해당 yaml파일에 새로운 commit이 생긴것을 확인

```
spec:
  containers:
  - name: front-app-user
    image: suhwan11/frontend-user:14
```

- 해당 yaml파일에 들어가보면 실제로 이미지가 교체된 것을 확인



- ArgoCD가 git에 변경사항이 생긴것을 감지하고 변경사항에 맞게 새롭게 배포한다.
- 새롭게 배포할때 새로운 replicaset을 만들고 그 아래 새로운 이미지가 적용된 컨테이너를 가진 파드를 Rolling Update를 통해 배포한다.

2.2.3.4. Frontend - Admin Pipeline 구축

- 위의 Pipeline들과 마찬가지로 플로우는 동일하다.
- 마찬가지로 인스턴스 환경에서 도커라이징 테스트 후 이미지 생성 및 컨테이너 배포가 정상적으로 확인되면 Pipeline 구축작업을 진행한다.

2.2.3.4.1. nginx에서 Frontend - Backend 연결

```
location /api {
    proxy_pass
http://k8s-default-backing-8476de3e44-889811917.ap-northeast-2.elb.amazo
naws.com;
    proxy_buffer_size   128k;
    proxy_buffers       4 256k;
    proxy_busy_buffers_size 256k;
  }
```

- nginx.conf 파일 생성 및 경로를 구분하여 Backend 주소를 등록한다.
- Admin 페이지에서는 동물 판독기 서비스는 지원하지 않기 때문에 Machine Learning Backend 주소는 등록하지 않는다.

2.2.3.4.2. Frontend 설정파일에서 Frontend - Backend 연결

```
server:{
      proxy:{
        "/api":{
          target:
"http://k8s-default-backing-8476de3e44-889811917.ap-northeast-2.elb.amaz
onaws.com/api/",
          rewrite: (path)=>path.replace(/^\/api/,""),
        },
      }
    }
```

- vite.config.js 파일 접속 및 경로를 구분하여 Backend 주소를 등록한다.
- Admin 페이지에서는 동물 판독기 서비스는 지원하지 않기 때문에 Machine Learning Backend 주소는 등록하지 않는다.

2.2.3.4.3. Frontend - Admin 도커라이징

```
FROM node:lts-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:stable-alpine
COPY ./nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=builder /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx",  "-g", "daemon off;"]
```

- 위와 같이 Dockerfile 작성 후 빌드를 통해 이미지를 생성하고, 컨테이너로 배포한다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

그림 67 Admin 로그인

- 컨테이너 배포 후 브라우저에 인스턴스 퍼블릭 IP 입력 시 위와같이 정상적으로 Admin페이지가 나타나는 것을 확인한다.

2.2.3.4.4. Jenkins를 이용한 Pipeline

```
    image: gcr.io/kaniko-project/executor:debug
    command: ['sleep']
    args: ['infinity']
    volumeMounts:
    - name: registry-credentials
      mountPath: /kaniko/.docker
  volumes:
  - name: registry-credentials
    secret:
      secretName: regcred
      items:
```

```
      - key: .dockerconfigjson
        path: config.json
'''
      }
    }

    stages{
        stage('checkout'){
          steps{
            container('gradle'){
              git branch: 'main',
url:'https://github.com/suhwan12/course-registration-GoormUniversity-adm
in.git'
              }
            }
        }
        stage('npm install & npm Build & Docker Build & Tag Docker
image'){
          steps{
            container('kaniko'){
                    sh "executor --dockerfile=Dockerfile \
                    --context=dir://${env.WORKSPACE} \
                    --destination=suhwan11/frontend-admin:latest \

--destination=suhwan11/frontend-admin:${env.BUILD_NUMBER}"
                }
            }
        }
        stage('Update K8s to New Frontend Deployment'){
          steps{
            container('gradle'){
                git branch: 'main' ,
url:'https://github.com/suhwan12/finalproject-argocd.git'
                sh 'sed -i "s/image:.*/image:
suhwan11\\/frontend-admin:${BUILD_NUMBER}/g"
front-deployment-admin.yaml'
                sh 'git config --global user.name suhwan12'
                sh 'git config --global user.email xman0120@naver.com'
                sh 'git config --global --add safe.directory
/home/jenkins/agent/workspace/Frontend-Pipeline-Admin'
```

```
                sh 'git add front-deployment-admin.yaml'
                sh 'git commit -m "Jenkins Build Number -
${BUILD_NUMBER}"'
                withCredentials([gitUsernamePassword(credentialsId:
'github-credentials', gitToolName: 'Default')]) {
                    sh 'git push origin main'
                }
            }
        }
    }
}
```

- 위의 Frontend - User Jenkinsfile과 마찬가지로 작성한다.
- 연동시킬 개발자 코드가 존재하는 git 이름이나, Docker Hub 레포지토리 이름, Sed명령어로 바꿀 yaml파일의 이름 등을 바꾸는 것에 유의한다.

2.2.3.4.4.1. Pipeline 결과 테스트

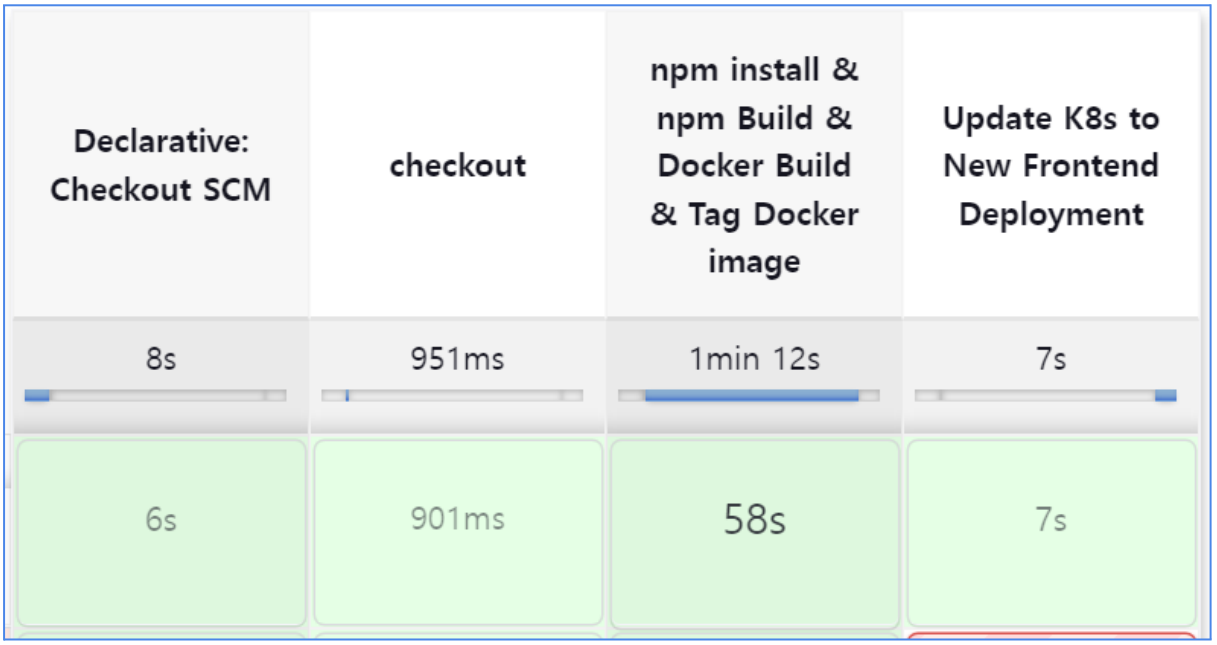
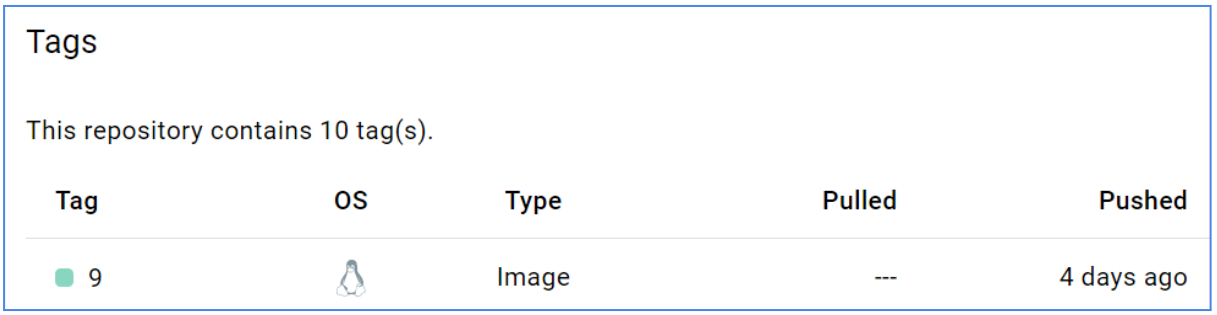


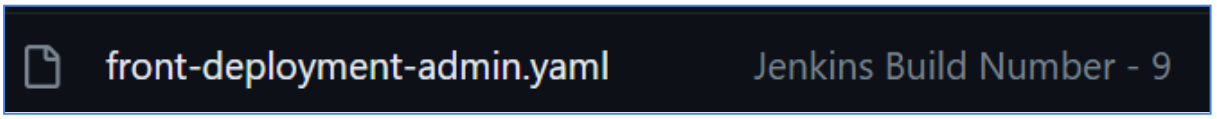
그림 68 *Pipeline 결과 테스트 2*

- Frontend - Admin Pipeline을 빌드하였을 때 정상적으로 빌드되는 것을 확인

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	



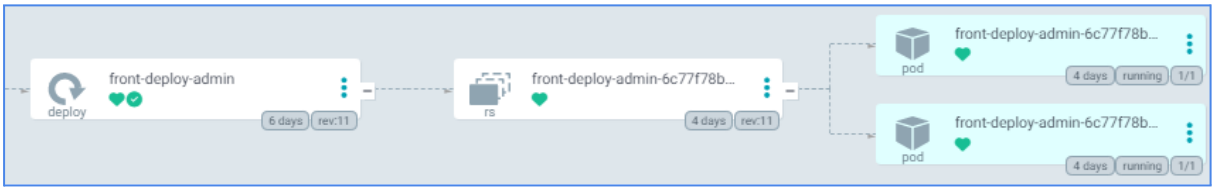
- Docker Hub에도 생성된 image가 Push된 것을 확인



- ArgoCD가 연동중인 git의 해당 yaml파일에 새로운 commit이 생성된 것을 확인

```
spec:
  containers:
  - name: front-app-admin
    image: suhwan11/frontend-admin:9
```

- 해당 yaml파일에 들어가보면 실제로 이미지가 교체된 것을 확인



- ArgoCD가 git에 변경사항이 생긴것을 감지하고 변경사항에 맞게 새롭게 배포한다.
- 새롭게 배포할때 새로운 replicaset을 만들고 그 아래 새로운 이미지가 적용된 컨테이너를 가진 파드를 Rolling Update를 통해 배포한다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2.4. Monitoring

2.2.4.1. Prometheus & Grafana

Prometheus는 오픈 소스 모니터링 시스템으로 다양한 서버, 서비스, 애플리케이션에서 발생하는 메트릭을 수집하고 저장한다. Grafana는 데이터 시각화 및 대시보드 툴이다. Prometheus와 Grafana를 통해 시스템 상태의 변화를 빠르게 파악하고 대응할 수 있도록, 효과적인 시스템 모니터링을 구축한다.

2.2.4.1.1. Prometheus & Grafana 배포

프로메테우스의 배치 위치

프로메테우스는 구성한 클러스터 내부 또는 외부에 배치할 수 있다. 현재 모니터링 해야 하는 EKS 클러스터의 규모는 그리 큰 규모가 아니다. 따라서 EKS 클러스터 내부에 배치하여 저렴한 데이터 전송 비용이라는 이점과 높은 모니터링 데이터 정확도를 갖도록 한다.

Prometheus, Grafana 설치

Prometheus와 Grafana는 kube-prometheus-stack을 통해 설치한다.

kube-prometheus-stack이란, 클러스터에서 모니터링을 구축하기 위한 프로메테우스와 그라파나를 포함하는 패키지이다. 이를 통해 클러스터에서 Prometheus 서버를 실행할 새로운 Pod를 배포한다.

```
helm repo add prometheus-community \
https://prometheus-community.github.io/helm-charts
helm repo update

# monitoring 네임스페이스 생성
kubectl create ns monitoring
helm install my-prometheus prometheus-community/kube-prometheus-stack -n
monitoring
```

코드 Prometheus, Grafana 설치

values.yaml 파일 커스텀

Helm 차트를 통해 설치할 때 설정할 수 있는 다양한 옵션 값들을 가진 values.yaml 파일을 수정한다. Prometheus의 서비스 유형을 LoadBalancer로 설정한 후 helm upgrade를 진행한다.

```
PS C:\Users\soo38> kubectl get svc -n monitoring
NAME                                      TYPE           CLUSTER-IP      EXTERNAL-IP
   PORT(S)                      AGE
alertmanager-operated                     ClusterIP      None            <none>
   9093/TCP,9094/TCP,9094/UDP   19h
prometheus-grafana                        LoadBalancer   10.100.85.108   a7189110be6ad44a8b4f2898838b4646-1227462100.ap-northeast-2.elb.amazonaws.com
   80:32671/TCP                 19h
prometheus-kube-prometheus-alertmanager   ClusterIP      10.100.89.89    <none>
   9093/TCP                     19h
prometheus-kube-prometheus-operator       ClusterIP      10.100.69.81    <none>
                        443/TCP                    19h
prometheus-kube-prometheus-prometheus     ClusterIP      10.100.39.71    <none>
   9090/TCP                     19h
prometheus-kube-state-metrics             ClusterIP      10.100.57.253   <none>
   8080/TCP                     19h
prometheus-operated                       ClusterIP      None            <none>
   9090/TCP                     19h
prometheus-prometheus-node-exporter       ClusterIP      10.100.74.110   <none>
                      9100/TCP                   19h
```

그림 69 서비스 확인

위의 모습과 같이 Prometheus와 Grafana에 필요한 서비스들이 성공적으로 설치된 모습을 확인할 수 있다.

2.2.4.1.2. Grafana 접속 및 Dashboard 구성

Grafana 접속

앞서 LoadBalancer로 지정한 외부 IP 주소를 통해 Grafana에 접속한다.

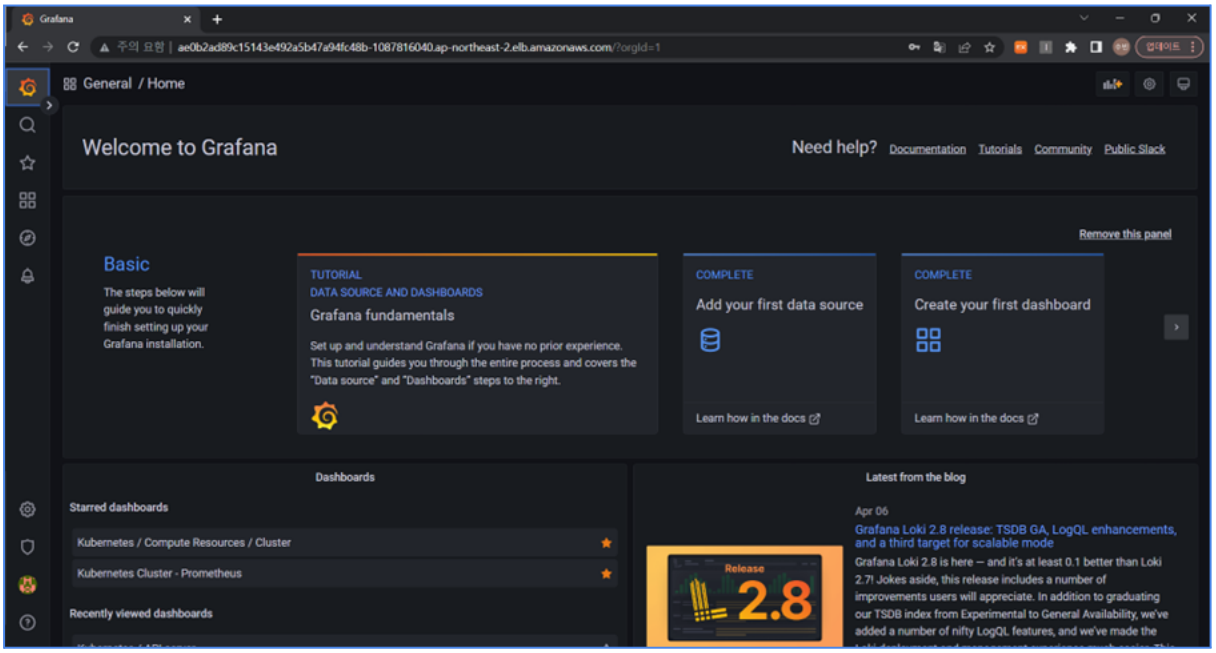


그림 70 Grafana webserver

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

Dashboard 구성

Prometheus에서 가져온 정보를 시각화하기 위한 대시보드를 구성한다. 대시보드를 구성하기 앞서 우선 Data source를 추가한다.

Grafana 홈 화면 – Add your first data source – Prometheus 이동

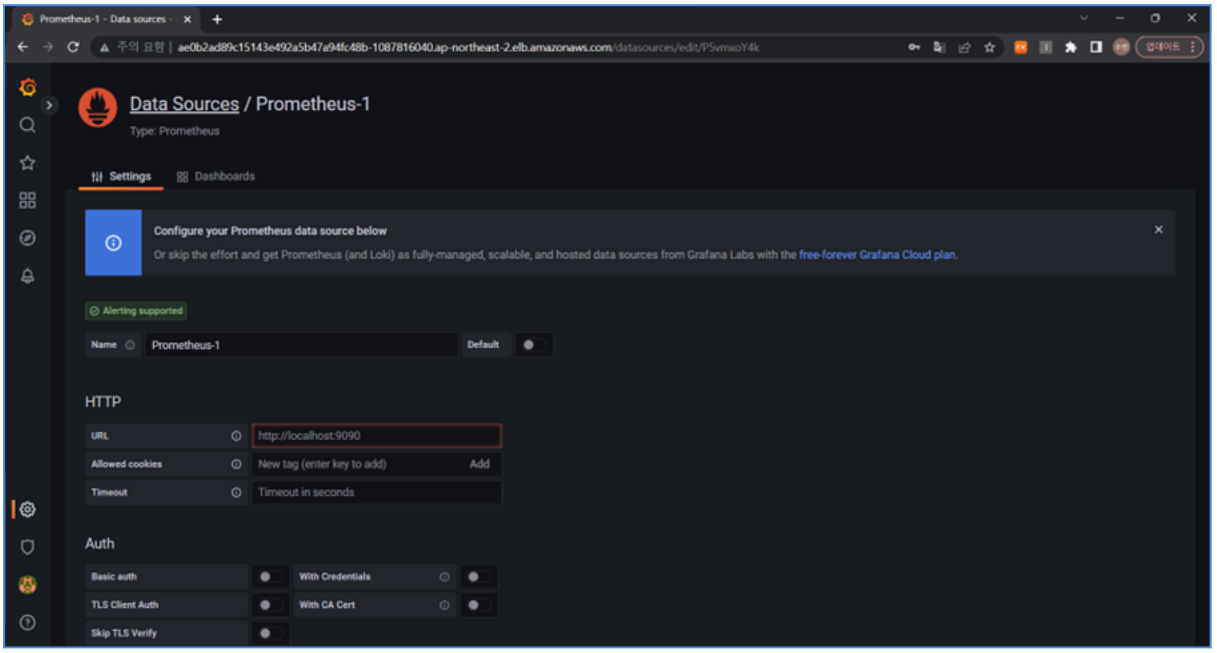


그림 71 Prometheus Data source 추가

- 이름을 설정하고 URL에는 my-prometheus-kube-prometh-prometheus 서비스의 주소를 http:// my-prometheus-kube-prometh-prometheus:9090 형태로 적어준다.
- my-prometheus-kube-prometh-prometheus는 프로메테우스 서버에 접근하기 위한 ClusterIP를 제공하며 9090/TCP포트를 통해 클라이언트에서 프로메테우스에 접근할 수 있도록 한다.
- 대시보드는 좌측의 DashBoards를 통해 다양한 리소스들을 시각화하여 확인할 수 있다. 이와 더불어 쿠버네티스 용 대시보드를 다운받아 설정한다.
- DashBoard – Import – Upload JSON file을 통해 다운 받은 파일을 업로드한다.

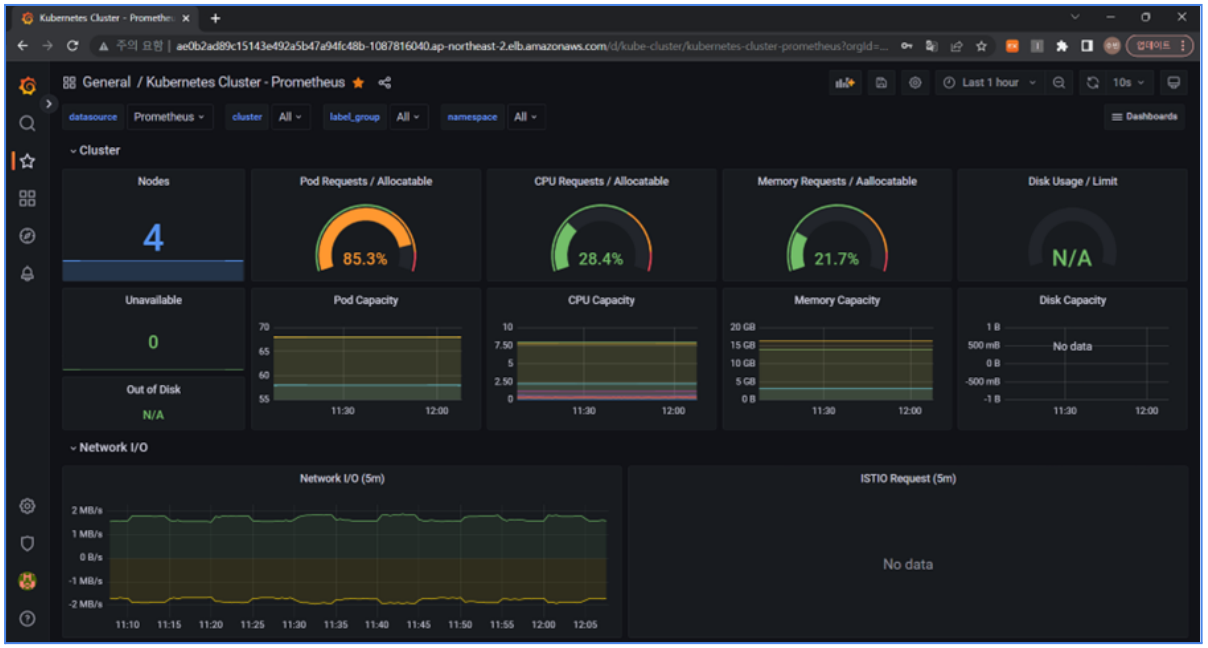


그림 72 Dashboard

2.2.4.2. Prometheus HA

모니터링에서도 고가용성을 확보하고자 한다. 첫 번째 시나리오로는 프로메테우스 서버를 두 개 이상으로 복제하여 사용하는 시나리오이다. 하지만 이 경우, 각각의 프로메테우스 서버에서 측정된 메트릭 데이터가 중복 저장되는 문제가 발생한다. 또한, Stateful 구성을 사용하므로 복제된 Pod들이 PVC를 공유할 수 없어 프로메테우스 서버 간의 데이터 공유 문제도 발생한다.

위와 같은 이유로 프로메테우스의 고가용성을 확보하기 위해 사용되는 오픈소스 툴인 Thanos를 이용하여 프로메테우스를 더 효율적으로 이용하고자 한다. Thanos는 프로메테우스 서버의 데이터를 중앙 집중화하여 저장하고, 이를 이용해 데이터를 쿼리하고, 분석하는 기능을 제공한다. 이로 인해 고가용성 및 분산 시스템에서의 지연 시간 최적화가 가능하다.

2.2.4.2.1. Object Storage 구성

Object Storage 사용 시 데이터 처리 및 보존성, 확장성, 비용 측면의 이점을 가진다. 따라서 로컬 저장소가 아닌 Object Storage를 통해 Prometheus 데이터를 저장한다. Thanos Object Storage로 많이 사용되는 MinIO를 Object Storage로써 이용한다.

- minio yaml파일 생성

```
curl
https://raw.githubusercontent.com/minio/docs/master/source/extra/examples/minio-dev.yaml -o minio-dev.yaml
```

코드 minio yaml파일 생성

해당 명령어를 통해 minio-dev.yaml파일이 생성된다. 해당 파일을 현재 프로젝트에 맞게 수정을 한다.

- minio-dev.yaml 수정 내용
  - Nodeselector 파트 삭제
  - Volume 설정 수정

```
# minio-dev.yaml



# Deploys a new MinIO Pod into the metadata.namespace Kubernetes
namespace
#
# The `spec.containers[0].args` contains the command run on the pod
# The `/data` directory corresponds to the
`spec.containers[0].volumeMounts[0].mountPath`
# That mount path corresponds to a Kubernetes HostPath which binds
`/data` to a local drive or volume on the worker node where the pod runs
#
apiVersion: v1
kind: Pod
metadata:
  labels:
      app: minio
  name: minio
  namespace: monitoring # Change this value to match the namespace
metadata.name
spec:
  volumes:
  - name: efs-volume
      persistentVolumeClaim:
      claimName: efs-pvc-monitoring
```

```
  containers:
  - name: minio
      image: quay.io/minio/minio:latest
      command:
      - /bin/bash
      - -c
      args:
    - minio server /data --console-address :9091
      volumeMounts:
      - mountPath: /data
      name: efs-volume
```

코드 minio-dev.yaml

수정한 minio-dev.yaml 파일의 내용은 위와 같다.

MinIO 사용 시 필요한 서비스 생성

MinIO의 볼륨은 현재 프로젝트에서 사용 중인 EFS로 지정한다. 하지만 기존의 EFS 서비스는 default 네임스페이스에 존재한다. 따라서 MinIO가 설치된 monitoring 네임스페이스에서 EFS를 사용할 수 있도록 efs-pvc-monitoring.yaml 이라는 이름의 PVC 생성 yaml 파일을 만들어 준다.

```
# efs-pvc-monitoring,yaml


apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-pvc-monitoring
  namespace: monitoring
spec:
  accessModes:
      - ReadWriteMany
  storageClassName: efs-sc
  resources:
      requests:
```

```
      storage: 1Gi
```

코드 efs-pvc-monitoring

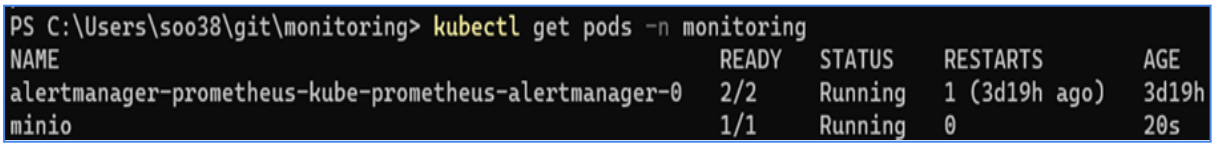


그림 73 MinIO pod 생성

minio-svc.yaml 작성

MinIO 웹페이지 연결을 위한 서비스를 생성한다.

```
# minio-svc.yaml


apiVersion: v1
kind: Service
metadata:
  name: minio-svc
  namespace: monitoring
spec:
  selector:
      app: minio
  ports:
      - protocol: TCP
      port: 9091
      targetPort: 9091
```

코드 minio-svc.yaml

minio-svc의 EXTERNAL-IP 주소를 통해 minio 사이트에 접속한다.

주소 : http://<External-IP>:9091

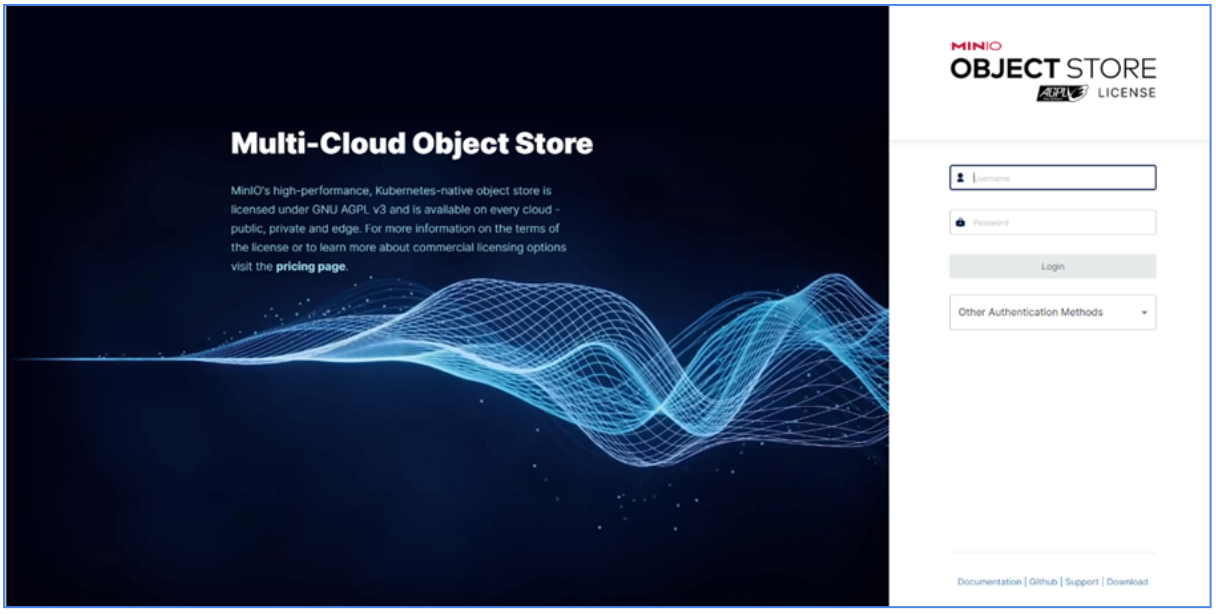


그림 74 MinIO 웹 페이지 접속

초기 아이디, 비밀번호인 minioadmin으로 로그인 후, MinIO의 버킷을 생성한다.

Thanos에서 사용될 버킷으로, thanos-bucket이라는 이름으로 생성해준다.

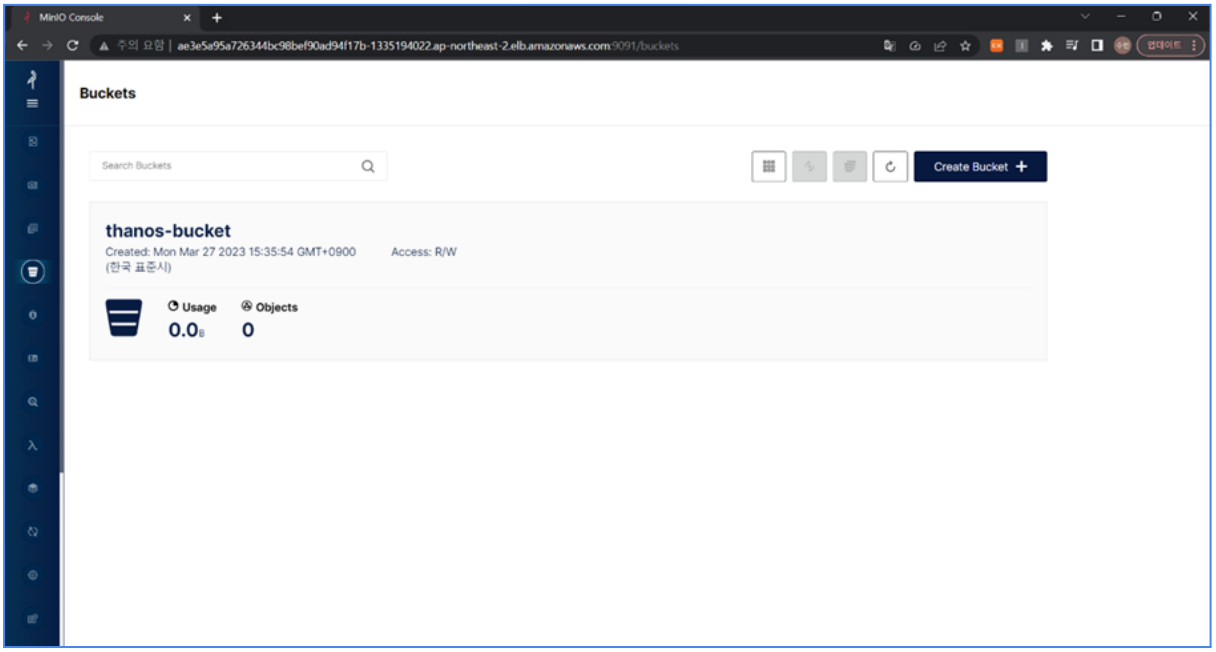


그림 75 MinIO 버킷 (thanos-bucket) 생성

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

2.2.4.2.2. Secret

MinIO에 연결하기 위한 시크릿을 작성한다. 이를 위해 우선 시크릿 파일을 작성한다.

```
# minio_conf.yaml

type: s3
config:
  bucket: "thanos-bucket"
endpoint:
"s3.ap-northeast-2.ae3e5a95a726344bc98bef90ad94f17b-1335194022.ap-northe
ast-2.elb.amazonaws.com"
access_key: "minioadmin"
  secret_key: "minioadmin"
```

코드 minio-conf.yaml

이를 통해 Prometheus와 Thanos의 컴포넌트에서 MinIO에 접근하기 위한 AWS S3 호환 엔드포인트를 설정한다. 해당 S3 엔드포인트를 통해 MinIO와 같은 다른 스토리지 시스템을 AWS S3와 같은 클라이언트로 사용할 수 있다.

위의 시크릿 파일(minio_conf.yaml)을 통해 thanos-obstore-secret-v1 이름의 시크릿을 생성한다.

```
kubectl create secret generic thanos-obstore-secret-v1 -n monitoring
--from-
```

코드 thanos-obstore-secret-v1 시크릿 생성

Monitoring 네임스페이스에 생성한다. 해당 시크릿을 참조하여 Thanos 구성파일에서 Minio 서버에 연결한다.

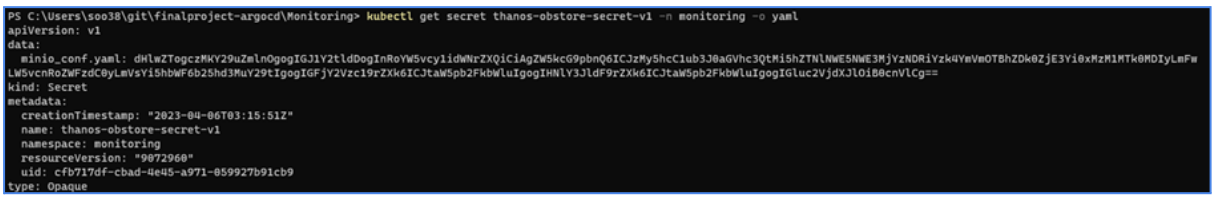


그림 76 thanos-obstore-secret-v1 시크릿 확인

2.2.4.2.3. Thanos sidecar 설정

Prometheus에서 Thanos를 사용하기 위해 values.yaml에 추가 설정을 진행한다.

values.yaml 수정

- prometheusOperator.thanos 추가 설정

```
thanos:
      image: "quay.io/thanos/thanos:v0.30.2"
      objectStorageConfig:
      key: "minio_conf.yaml"
      name: "thanos-obstore-secret-v1"
      version: v0.30.2
```

코드 values.yaml - thanos 설정 추가

- thanosService : true
- thanosService.type : LoadBalancer
- thanosService.clusterIP : “”
- prometheusSpec.externalUrl : "http://my-prometheus-kube-prometh-prometheus.monitoring.svc.cluster.local"

위와 같이 코드를 추가 및 수정한다. 해당 설정들을 완료하면 thanos-sidecar인 my-prometheus-kube-prometh-thanos-discovery 서비스가 생성된다.

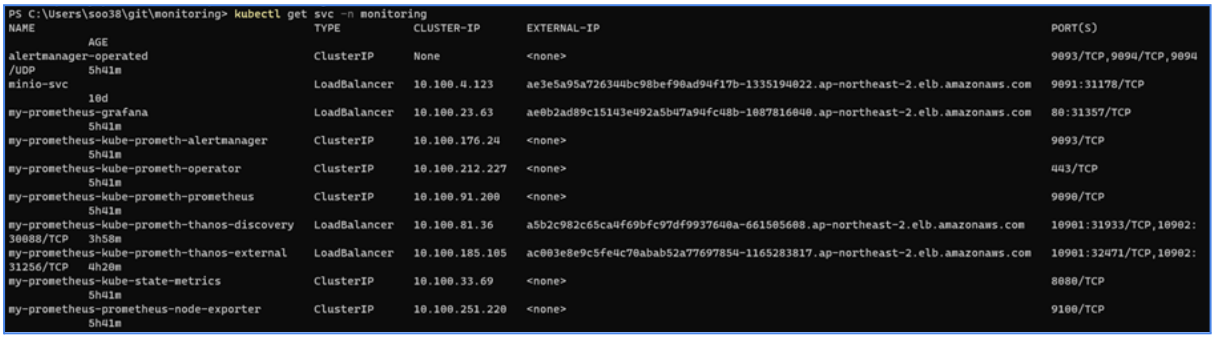


그림 77 Thanos sidecar 생성 (my-prometheus-kube-thanos-discovery)

2.2.4.2.4. Thanos 설치

```
# local path 설치
```

```
kubectl apply -f
https://raw.githubusercontent.com/rancher/local-path-provisioner/master/
deploy/local-path-storage.yaml
```

코드 local path 설치

Thanos 설치 시 필요한 내용들을 추가한 사용자 정의 Thanos Helm Chart 구성 파일(thanos.yaml)을 작성한다.

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install thanos bitnami/thanos --version 12.3.1 -f thanos.yaml -n
monitoring
```

코드 thanos.yaml

위의 thanos.yaml 과 함께 Thanos를 설치한다.

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install thanos bitnami/thanos --version 12.3.1 -f thanos.yaml -n
monitoring
```

코드 Thanos 설치

포트포워딩을 통해 thanos-query 웹 사이트에 접속한다.

```
export SERVICE_PORT=$(kubectl get --namespace monitoring -o
jsonpath="{.spec.ports[0].port}" services thanos-query)
```

코드 포트포워딩

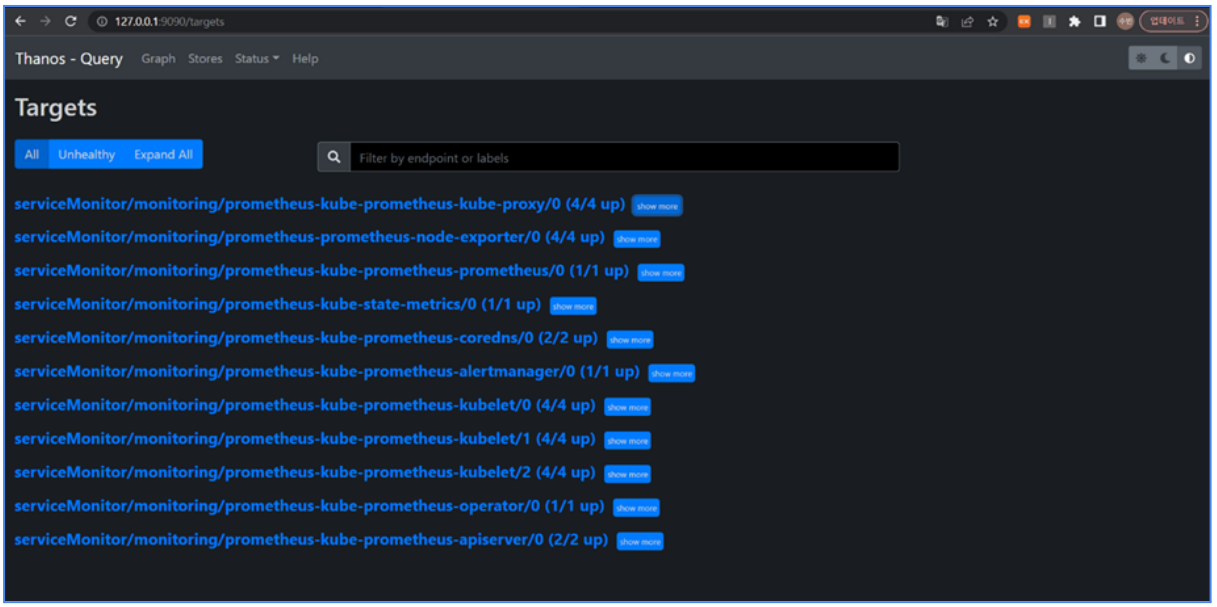


그림 78 thanos-query의 targets 확인

Thanos – Query 웹 사이트에 정상적으로 접속되는 모습을 확인할 수 있다. 더불어 targets에 프로메테우스에서 받아온 메트릭들을 성공적으로 받아오고 있다.

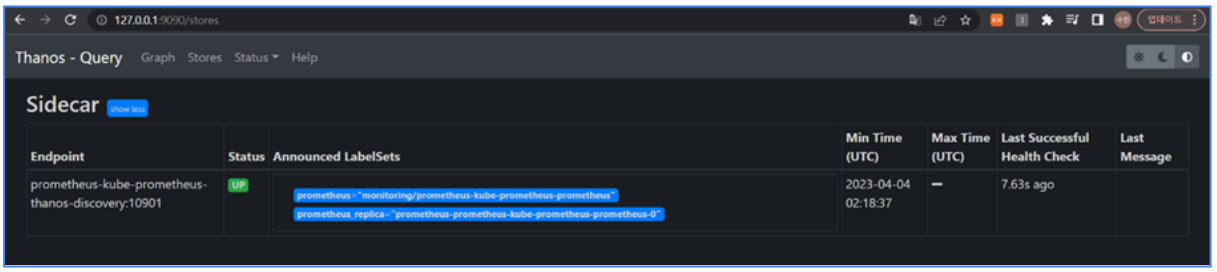


그림 79 thanos-sidecar 확인

다음과 같이 thanos-sidecar 또한 정상적으로 동작하고 있는 것을 확인할 수 있다.

2.2.4.2.5. Prometheus HA 구성

Thanos까지 정상적으로 구현되었으므로 본격적인 Prometheus HA를 구성한다.

이중화를 위해 Prometheus 서비스를 2개로 늘려준다.

- Values.yaml의 prometheusSpec.replicas : 2로 수정

```
PS C:\Users\soo38\git\monitoring> kubectl get pod -n monitoring
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-my-prometheus-kube-prometh-alertmanager-0   2/2     Running   0          6h28m
my-prometheus-grafana-6d46f47796-vd4cb                   3/3     Running   0          6h28m
my-prometheus-kube-prometh-operator-54bcf66c88-rvj9c     1/1     Running   0          6h28m
my-prometheus-kube-state-metrics-6cf7fbdcb5-cwrhr        1/1     Running   0          6h28m
my-prometheus-prometheus-node-exporter-k9dwf             1/1     Running   0          74m
my-prometheus-prometheus-node-exporter-nb7td             1/1     Running   0          78m
my-prometheus-prometheus-node-exporter-tt65m             1/1     Running   0          74m
my-prometheus-prometheus-node-exporter-xh4fv             1/1     Running   0          78m
prometheus-my-prometheus-kube-prometh-prometheus-0       3/3     Running   0          5h7m
prometheus-my-prometheus-kube-prometh-prometheus-1       3/3     Running   0          64m
thanos-query-54db769888-mrqbx                            1/1     Running   0          63m
thanos-query-frontend-86957c5694-f9tzf                   1/1     Running   0          93m
thanos-ruler-0                                           1/1     Running   0          93m
```

그림 80 2개의 prometheus 서비스 생성

위와 같이 기존에 하나였던 Prometheus 서비스가 prometheus-my-prometheus-kube-prometh-prometheus-0, prometheus-my-prometheus-kube-prometh-prometheus-1 2개로 늘어난다.

추가로 생성된 Prometheus 서비스(prometheus-my-prometheus-kube-prometh-prometheus-0, prometheus-my-prometheus-kube-prometh-prometheus-1) Pod를 Thanos와 연결하기 위한 서비스를 생성한다. Pod 별로 서비스를 각각 생성할 시, 클러스터내의 서비스 수 증가로 불필요한 리소스가 소모된다. 따라서 headless 서비스를 생성하여 각각의 RelicaSet을 하나의 headless 서비스에 연결한다.

Headless 서비스 생성

```
  app: prometheus
clusterIP: None
ports:
  - name: prometheus
    port: 9090
    targetPort: 9090
```

코드 prometheus-headless-svc.yaml

다음과 같이 prometheus-headless-svc에서 연결할 Pod에 대한 label 셀렉터로 prometheus를 설정했다. 따라서 Prometheus 서비스(prometheus-my-prometheus-kube-prometh-prometheus-0, prometheus-my-prometheus-kube-prometh-prometheus-1) 가 prometheus 라벨 값을 갖도록 설정한다.

label 설정

```
kubectl label pod prometheus-my-prometheus-kube-prometh-prometheus-0 -n
monitoring app=prometheus

kubectl label pod prometheus-my-prometheus-kube-prometh-prometheus-1 -n
monitoring app=prometheus
```

코드 prometheus label 설정

다음으로 thanos.yaml의 querier.stores부분에 위에서 설정한 prometheus-headless-svc를 추가한다.

thanos.yaml 추가 설정

```
  app: prometheus
clusterIP: None
ports:
  - name: prometheus
    port: 9090
    targetPort: 9090
```

코드 querier.stores 주소 추가

Query 서비스는 Prometheus와 통신 시 gRPC 주소를 사용한다. 따라서 stores에 Thanos Query 서비스가 Promethues에 접근할 수 있도록 gRPC 주소를 설정해준다.

그 후 수정된 thanos.yaml파일을 가지고 helm upgrade를 진행한다.

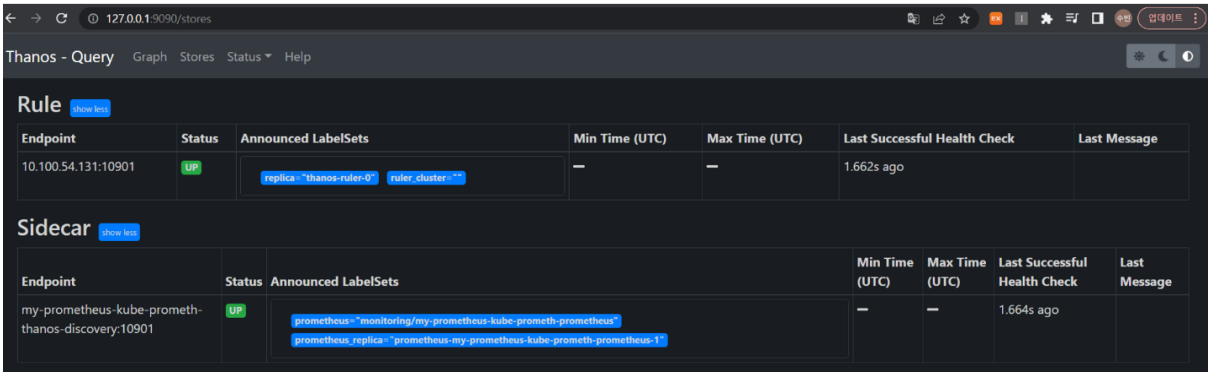


그림 81 Prometheus HA 설정 후 Sidecar 모습(1)

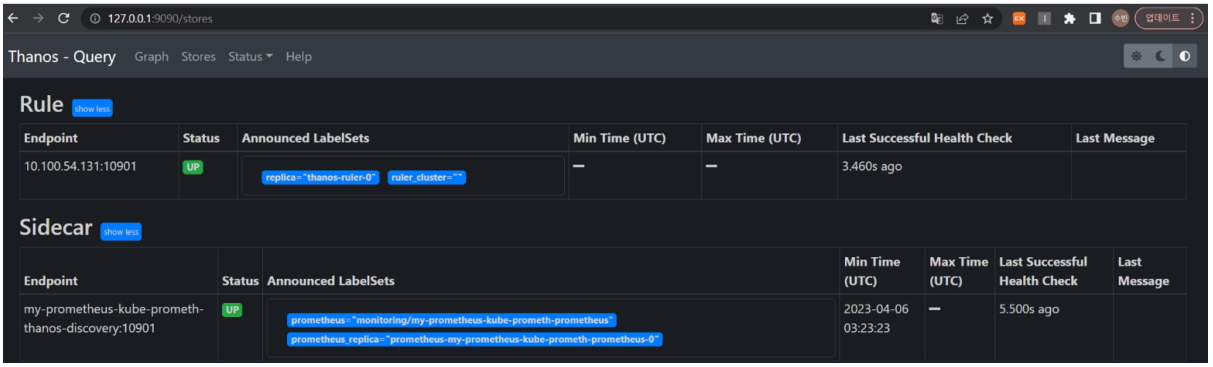


그림 82 Prometheus HA 설정 후 Sidecar 모습(2)

다음과 같이 수정된 thanos.yaml이 정상적으로 반영되었다. Sidecar 모습(1)과 같이 prometheus-my-prometheus-kube-prometh-prometheus-1을 이용하는 와중, 새롭게 Thanos를 다시 시작했을 때 prometheus-my-prometheus-kube-prometh-prometheus-0을 이용하는 것을 확인할 수 있다. 이처럼 prometheus-my-prometheus-kube-prometh-prometheus-0과 prometheus-my-prometheus-kube-prometh-prometheus-1을 번갈아가며 사용하는 모습을 확인할 수 있다.

2.2.4.3. Amazon CloudWatch

2.2.4.3.1. Amazon CloudWatch 대시보드 구성

EKS 클러스터 및 AWS에서 사용하고 있는 리소스들의 상태를 실시간으로 모니터링 하기 위해 Amazon CloudWatch 웹 콘솔에 접속한다.

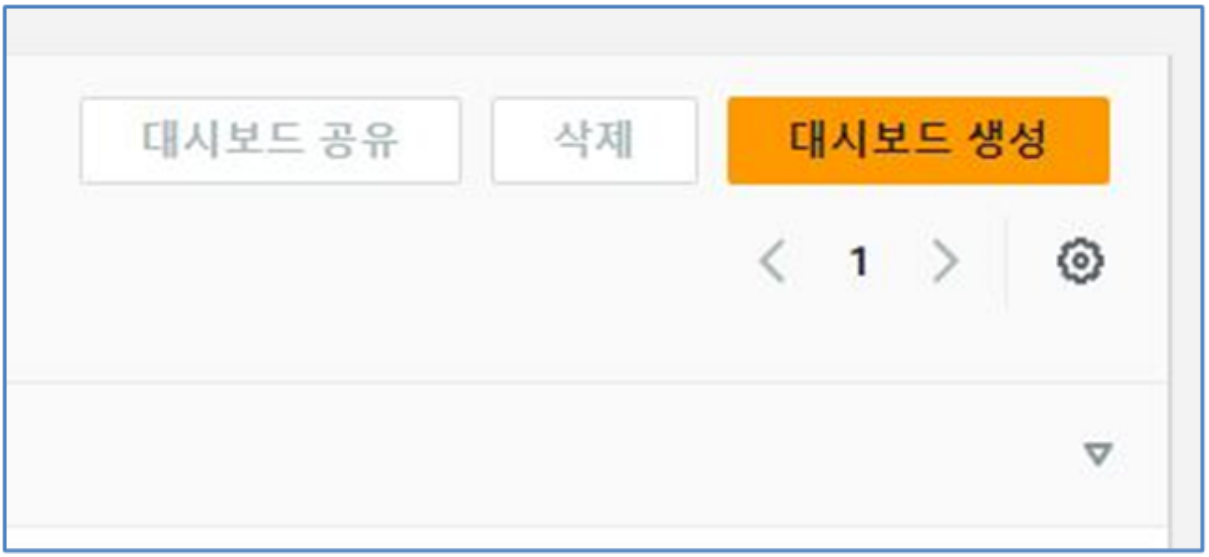


그림 83 대시보드 생성

Amazon CloudWatch 웹 페이지에 접속한 후, 오른쪽 상단에 보이는 '대시보드 생성' 버튼을 눌러 대시보드 생성 절차에 들어간다.

그림 84 대시보드 이름 지정

대시보드 생성 버튼을 클릭하면, 대시보드 이름을 지정하는 창이 출력되고, 여기서 새로 생성할 CloudWatch 대시보드의 이름을 규칙에 맞추어 지정해 준다. 대시보드에 원하는 이름을 넣은 다음 대시보드 생성 버튼을 누르면 대시보드가 생성이 되며, 생성된 대시보드는 내부에 어떤 요소도 표함되어 있지 않는 순정 상태의 대시보드로 만들어진다.

그림 85 대시보드에 포함할 위젯 생성

새로 생성된 대시보드에 처음으로 접근하면, 대시보드에 추가할 위젯을 선택하라는 창이 뜨고, 본인이 모니터링 하고자 하는 지표에 맞는 유형의 위젯을 선택하면 된다..

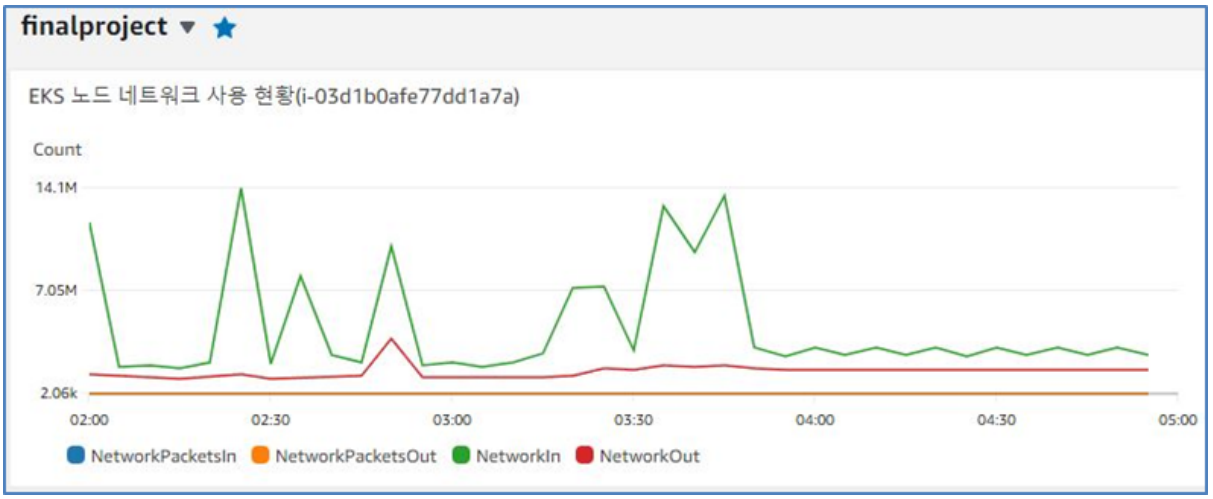


그림 86 '행' 위젯으로 볼 수 있는 EKS 노드의 네트워크 사용 현황

위 그림은 현재 작동하고 있는 EKS 클러스터의 노드들 중 한 노드의 네트워크 사용 현황을 '행' 위젯을 통해 모니터링 하도록 설정한 모습이다., 위젯 배치와 동시에 정해진 시간에 따라 어떻게 네트워크 사용량이 변화하는지 살펴볼 수 있도록 하였다.

그림 87 EKS 클러스터 노드들의 개별 EBS 볼륨 사용 현황

'행' 위젯 말고도 값을 표시할 수 있는 위젯은 다양하다. 위 그림에서 사용된 위젯은 '게이지' 위젯으로, 실시간 사용량, 점유율과 같은 값들을 사용량/여유 공간의 형식으로 시각화해주는 위젯이다. '게이지' 위젯을 사용하기 위해서는 별도로 게이지의 최소값과 최대값을 옵션 탭에서 지정해 주어야 한다.

그림 88 '게이지' 위젯 옵션 중 '게이지 범위'

현재 '게이지' 위젯에서 표시되도록 설정한 값은 최소가 0, 최대를 10000으로 표시해 두었다. 표시될 수 있는 값으로는 용량, 횟수 등 다양한 값이 존재한다.

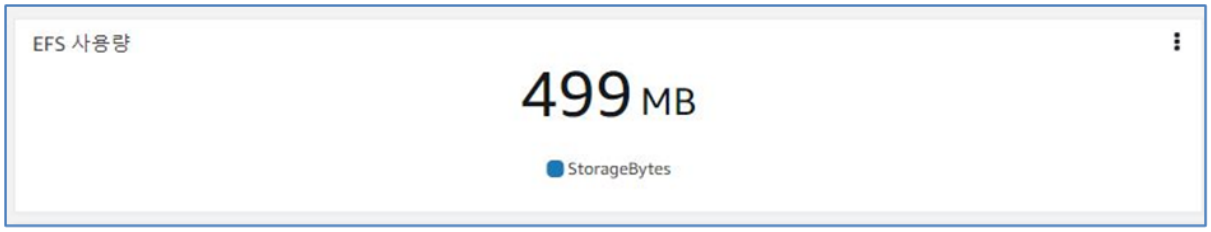


그림 89 '번호' 위젯으로 조회되는 Amazon Elastic File System(EFS)의 사용량

Amazon Elastic File System(EFS)의 사용량을 조회할 수 있도록 설정해 두었고, 사용한 위젯은 '번호' 위젯을 사용하여 얼마나 사용하였는지를 숫자로 표시하도록 해두었다.

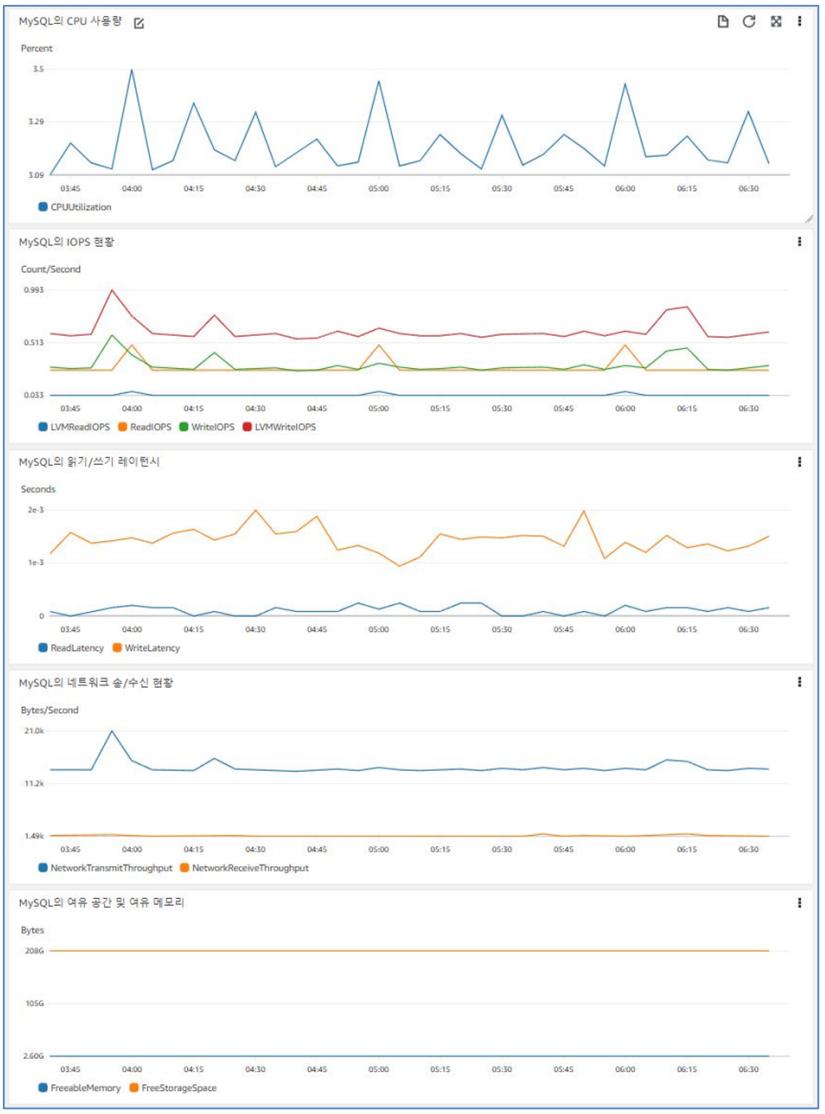


그림 90 '행' 위젯으로 표시되고 있는 RDS 인스턴스 내 MySQL의 상황

또한 현재 AWS 상에서 존재하며, 개발에 사용될 목적으로 배치되어 있는 RDS 인스턴스의 상황도 '행' 위젯으로 하여금 실시간 모니터링을 할 수 있도록 배치해 두었다. 현재 표시되는 값은 각각 RDS 인스턴스의 CPU 사용량, IOPS 현황, 레이턴시, 네트워크 송/수신 현황, 여유 공간 및 여유 메모리와 같은 값들이 실시간으로 표시되고 있다.

그림 91 표시 시간 단위

현재 실시간으로 지표 수집이 진행되고 있고, 현재 축적된 값들을 3시간 단위로 볼 수 있도록 설정해놓은 상태이다. 대시보드에 접속하면 보이는 화면 중 오른쪽 상단에 위치해 있는 시간을 조정하면 축적된 값들을 해당 시간에 맞추어 볼 수 있다.
예를 들어, 축적된 값들을 1시간 단위로 측정된 값을 보고 싶다면 '1시간' 버튼을 눌러 클릭하면, 즉시 1시간 동안 축적된 값을 보여주도록 설정할 수 있다. 그 외에도 본인이 원하는 시간 단위를 직접 지정하여 살펴볼 수 있다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

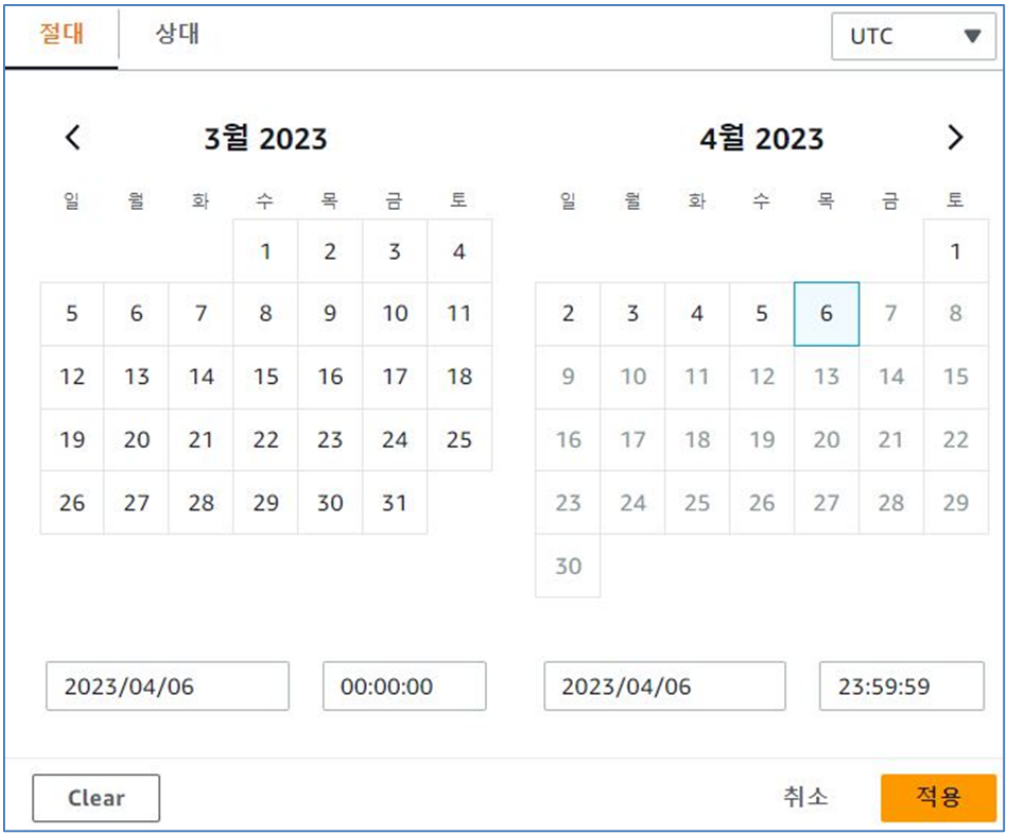


그림 92 원하는 시간대를 직접 지정하여 값을 조회하는 방법

원하는 값을 직접 지정하여 보는 방법으로는 시간 탭에서 '사용자 지정' 버튼을 누른 뒤 나오는 창에서 '절대'를 누르고 원하는 시작 시간과 끝 시간을 정해준 다음 출력되는 값을 조회하면 된다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

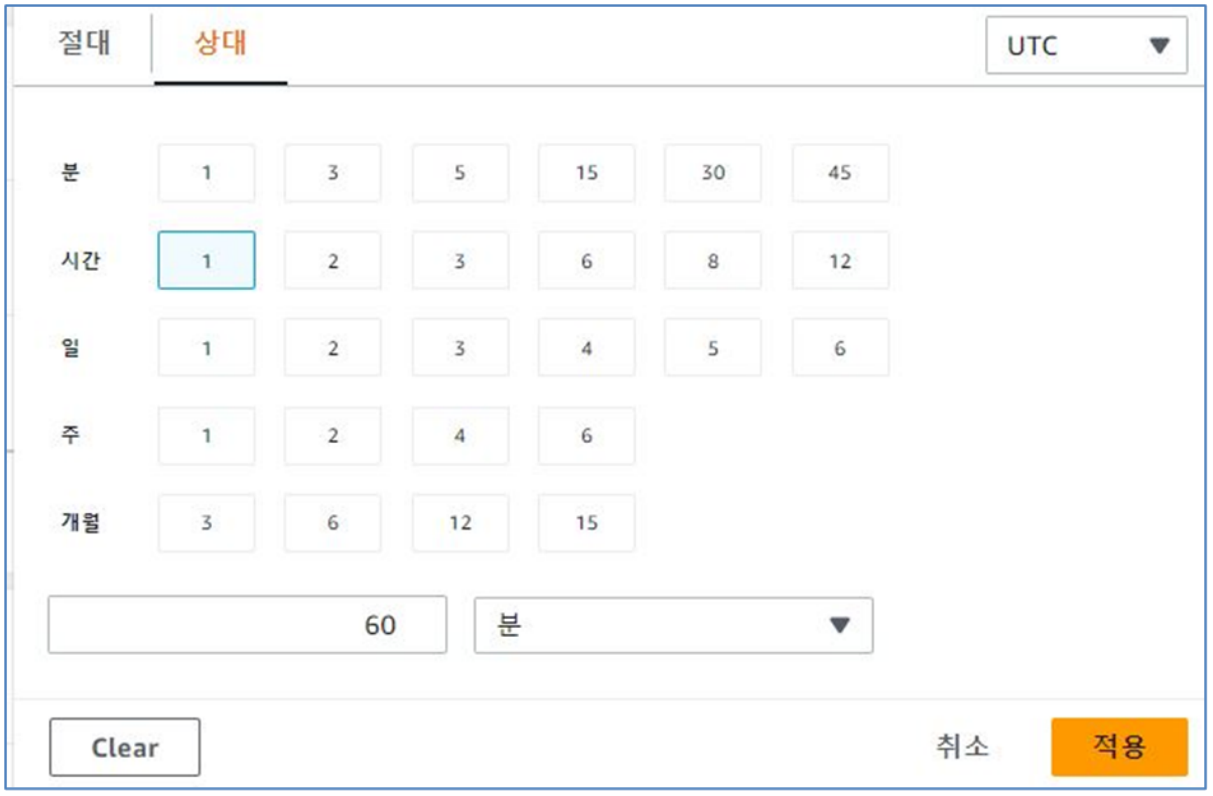


그림 93 현재 시간을 기준으로 조회할 값들을 정하는 방법

앞의 방법이 시간과 시간을 직접 정한 다음 값을 조회하는 방법이었다면, 위 그림의 방법은 현재 시간을 기준으로 얼마나 전 시간대의 값을 조회할 것인지 직접 지정해줄 수 있고, 특히 월 단위의 값까지 직접 지정할 수 있다는 것이 특징이다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

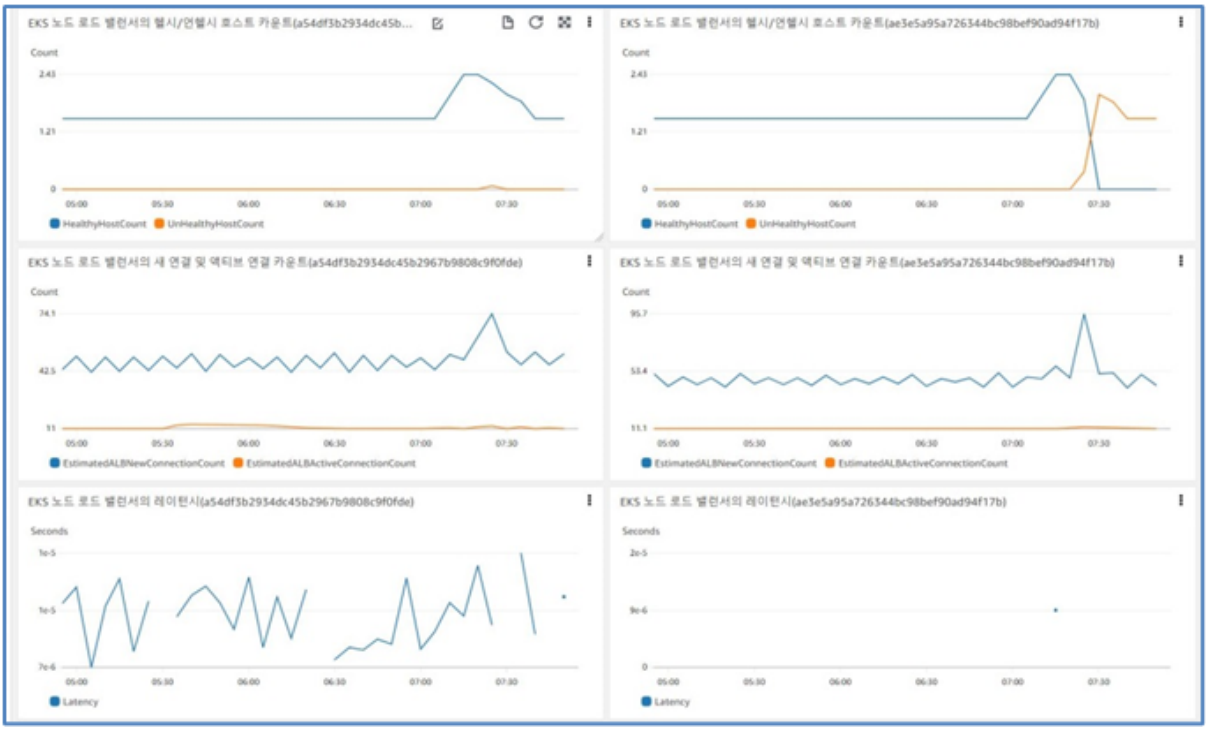


그림 94 EKS 노드 로드 밸런서의 지표 현황

또한 EKS 클러스터 내부의 서비스를 외부로 노출시키는데 사용되는 로드 밸런서들의 현황을 모니터링 하기 위해 각각 로드 밸런서의 헬시/언헬시 호스트 카운트, 새 연결 및 액티브 연결 카운트, 레이턴시를 표시하도록 설정해 두었다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

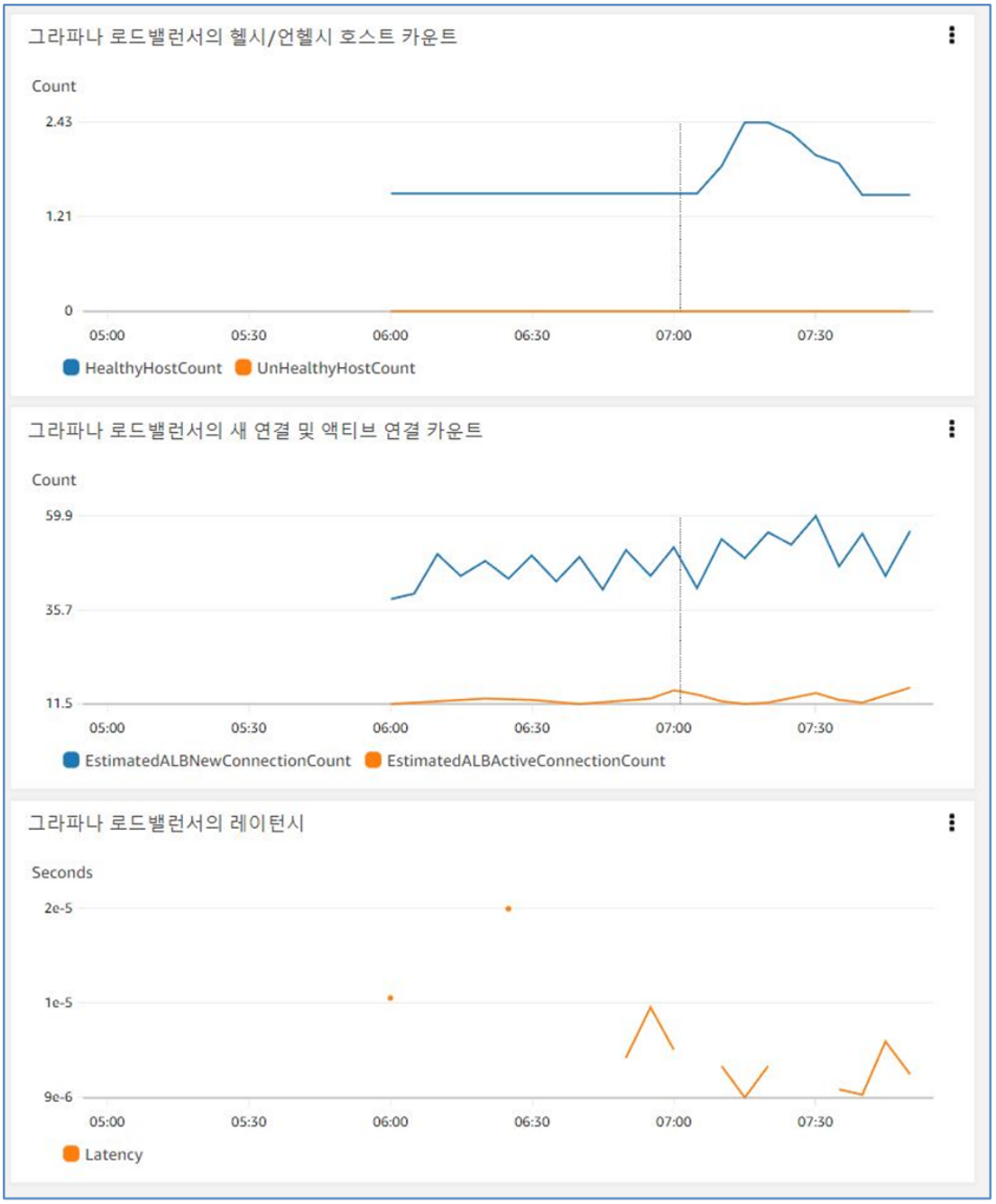


그림 95 그라파나 로드 밸런서의 지표 현황

또한 현재 EKS 클러스터 내부의 리소스들을 확인하기 위해 사용되는 툴인 그라파나(Grafana) 역시 로드 밸런서를 통해 접근할 수 있는 상태인데, 이때 사용되는 로드 밸런서를 모니터링 하기 위해 EKS 노드에서 사용되는 로드 밸런서의 모니터링 방법과 동일하게 위젯을 생성해 주었다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

3. 결론

이번 ***G3M*** 팀의 프로젝트는 성공적이라고 할 수 있다***.***

그 이유로는 목표했었던 체크리스트 구현을 전부 완료를 했기 때문이다.
목표했던 체크리스트는

- ☑ 학생의 수강신청 및 취소 기능의 구현
- ☑ 수강신청의 시간에 맞춰 수강신청이 열리고 닫히는 기능의 구현
- ☑ EKS를 사용한 쿠버네티스 인프라를 구축하여 고가용성 서비스 구현
- ☑ ArgoCD를 통한 자동화 배포
- ☑ Jenkins를 통한 자동화 빌드
- ☑ Prometheus, Grafana를 통한 Monitoring System 구축

이고, 기본적인 구현이 빠르게 끝나서 더 진행한 부분은

- ☑ AI MSA 구현
- ☑ 도메인 구현
- ☑ Amazon Cloud Watch를 통한 AWS 서비스 Monitoring System
- ☑ Thanos를 통한 모니터링 서비스의 고가용성 구현
- ☑ Monitoring System을 기반으로 AutoScale 구성
- ☑ Kaniko로 docker 없이 컨테이너 이미지 빌드

등을 더 진행하게 되었고 내부 테스트 결과 100명의 인원의 동시 사용에 무리가 없었고,
목표로 했던 프로젝트의 최중요 과제인

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

라는 과제를 훌륭하게 수행을 했다고 할 수 있다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

4. 추후과제 및 후기

4.1. 추후 과제

전체적인 서비스가 무거움

이유중 가장 큰 것은 **d** 개선이 필요할 것이다.

그다음 이유로는 **DB**의 최적화가 되어있지 않아서이다.
DB의 구조 자체도 개선 가능성이 있지만 더 중요한것은 백엔드단에서 API를 받아오는 행위에서 많은 데이터베이스의 조회가 발생하는데 시간관계상 N+1 문제가 나는 코드를 억지로 고쳐서 사용을 했기에 이것을 개선하면 API의 속도가 많이 개선되어 서비스의 속도를 많이 개선할 수 있을 것이다.

서비스 ***Page*** 디자인 개선

목표했던 과제인 인프라와 서비스를 구현하였지만 실 사용자의 경험을 개선하는것은 어떠한 서비스든 꼭 필요한 부분이라고 할 수 있다. 추후 디자이너와의 협업을 통하여 서비스페이지 디자인을 사용자 경험에 맞추어 개선을 할 수 있을것이다.

Machine Learning 서버 파이프라인 구축

Machine Learning 서버를 위한 Deployment도 EKS 클러스터에서 배포를 하였다. 하지만 시간 관계상 해당 서버를 배포하기 위한 빌드 테스트 **,** 도커라이징만을 진행하여 이미지 생성만 했기 때문에 파이프라인을 구축하지 못하였다. Machine Learning 서버도 파이프라인으로 구축함에 따라 추가 개발 및 유지보수 **,** 버전 관리를 용이하게 할 수 있을 것이다.

AWS CloudWatch 지표 추적 및 갱신 자동화 고안

현재 EKS 클러스터 노드들은 수요에 따라 자동으로 증가하거나 감소되는 상황이다. 그러나 AWS CloudWatch에서 추적 대상들 중, 추적 대상을 클러스터 노드로 잡아놓은 위젯들은 추적 대상인 노드가 종료되게 되면 추적 대상을 잃고 값을 표시하지 않게 되는데, 이때 다시 추적할 대상을 수동으로 일일이 지정해 주어야 하는 문제가 있다.
따라서 기존에 추적 중인 노드가 종료되고, 새 노드가 다시 생성되었을 때, 추적할 대상을 잃은 위젯들이 새 노드의 값을 자동으로 다시 추적하여 위젯으로 표시하도록 하는 방법을 고안할 필요가 있다.

Thanos를 이용한 완벽한 ***Prometheus HA*** 구현

Prometheus HA를 구현하기 위해 Thanos를 이용했다. Thanos sidecar를 구현하고 Prometheus와 Thanos를 성공적으로 연결하여 Prometheus 서비스들을 돌아가면서 사용하는 것 까지는 확인을 했다. 하지만 중복 메트릭 제거등 Thanos의 기능을 완전히 이용하여 완성도있는 고가용성을 구현하지 못하였다. 해당부분을 보완한다면 많은 양의 데이터를 모니터링해야하는 상황에서 더욱 완성도 있는 모니터링 고가용성을 갖출 수 있을 것이다.

4.2. 후기

정한교: 팀장으로 많은 부담이 있었는데 팀원들이 다들 잘 따라와 준데다가 프로젝트 결과도 꽤 만족스러워서 너무 좋았다. 메인 개발자로써 프로젝트에 참여하면서 인프라 엔지니어들과 협업하는 경험은 인프라엔지니어로 커리어를 생각하고 있는 나에게 데브옵스 문화를 개발자입장에서 반대로 생각을 해보게 되는 이런 경험은 어디에서 할 수 없는 경험인것 같아서 신선했다.

최수환: 직접 EKS 클러스터를 배포하고 , 설계한 인프라 아키텍처를 기반으로 클러스터 오브젝트를 작성해 배포해봤다. 또한 개발자의 코드에 대한 자동화 CI/CD 파이프라인을 구축하여 유지보수 및 버전관리를 가능하게 하였다. 이 프로젝트를 통하여 조금 더 인프라 엔지니어에 가까운 경험을 할 수 있어서 값진 경험이였다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

신현식: 인프라 구축부터 CI/CD 파이프라인까지 설계부터 시작하여 직접 구현까지 해본 경험이 매우 값진 것 같다. 이번 프로젝트를 하면서 공식문서도 많이 찾아보았고 팀원들의 도움도 많이 받았다. 이론으로만 배웠던 것들을 실제 프로젝트에 진행해보니 어려웠던 점도 많았고 한가지 오류를 잡기위해 하루종일 자료를 찾은 경험도 했었다. 쿠버네티스를 배우면서 이를 직접 적용하여 프로젝트를 진행하기에는 내가 아직 부족한 점이 많다고 느꼈다. 그래도 그렇게 고민한 끝에 결국 문제를 해결하고 내 지식으로 만든 경험이 나를 성장시켜주었고 인프라에 대한 전반적인 이해를 올려주는 좋은 경험이였다. 다음 프로젝트에서는 이번에 사용해보지 않았던 ECS와 Git Action를 활용하여 인프라 구축과 Ci/CD를 짜보고 싶다는 생각도 하였다.

배수빈: 이번 프로젝트를 통해 항상 관심만 가지고 있었던 인프라 분야를 직접 구현해 볼 수 있었던 좋은 경험이었다. 모니터링의 고가용성을 위해 생소한 Thanos와 MinIO를 이용하며 정말 많은 오류와 시행착오를 겪었다. 하지만 계속 원인을 파악해보고 다양한 시도를 해보며 해결할 수 없을 것 같던 오류들을 해결하고 성공적으로 프로젝트에 Thanos를 적용시킬 수 있어서 뿌듯했다. 다음에는 개발과 좀 더 가까운 CI/CD부분과 Terraform 사용에도 도전해보고싶다. 팀원들에게도 많이 배우며 인프라 분야에 좋은 첫 발걸음을 내딛을 수 있었다.

이승엽: 이론 및 실습 시간 때 잠깐 짚고 넘어갔던 프로메테우스 및 그라파나를 직접 다뤄볼 수 있는 기회가 생겨 좋았고, 또한 프로메테우스 뿐만 아니라 프로메테우스와 연동할 수 있는 Thanos, MinIO와 같은 다양한 요소들이 존재한다는 것을 알게 되어 여태까지 알고 있었던 지식에 층을 더할 수 있어서 좋은 시간이었다. 그리고 강의 시간에 다루지 않았던 여러 요소들을 직접 사용자 본인이 사용법을 학습하고 직접 다뤄볼 수 있었다는 것에 대해 아주 귀중한 시간이었다고 생각한다. 좋은 팀원들과 함께할 수 있어서 좋았다.

	Open			
	Category	첨부파일 버전	문서 최종 수정일	G3M
	Manual + Utility	1.1	2023.04.07	

5. 참고 자료

Vue 공식 문서
https://vuejs.org/

Helm 공식 홈페이지
https://helm.sh/

Prometheus 공식 홈페이지
https://prometheus.io/

쿠버네티스 공식 문서
https://kubernetes.io/docs/

AWS 공식 문서
https://docs.aws.amazon.com/ko_kr/

Kube-prometheus-stack GitHub
https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack

Grafana 대시보드 템플릿 홈페이지
https://grafana.com/grafana/dashboards/