스킬(기술)

SPRING BOOT

FLASK

MYSQL

ORACLE

AWS

DOCKER

GIT

REDIS

FLASK

PYTHON

링크

포트폴리오(**Vue.js** 기반)**:**
https://dongkyu.shop

Git:
https://github.com/dongkyukim1

블로그**:**
https://begin-developer.tistory.com/

언어

KOREAN

ENGLISH

김동규
1993.01.05

백엔드 개발자

서울특별시 양천구 신정3동 푸른마을 아파트 402동 401호, 서울, 08053, 대한민국

010-4048-8292

wlsntus55@gmail.com

자기 소개

Spring Boot 와 Flask 를 활용한 웹 애플리케이션 개발에 전문성을 갖춘 백엔드 개발자입니다.
Git 을 통한 형상 관리와 Docker 와 AWS 로 배포 경험이 있으며, LLM 를 활용한 AI 기능 구현 능력을 보유하고 있습니다.
Spring Boot와 Flask를 활용한 웹 애플리케이션 개발에 대한 깊은 전문성과 실무 경험을 갖춘 백엔드 개발자입니다.
Git을 활용한 체계적인 형상 관리 경험과 Docker 및 AWS를 통한 효율적인 배포 능력을 보유하고 있습니다. 최근에는 LLM을 활용한 AI 기능 구현에 도전하여, 기술적 깊이를 더하고 있습니다.
팀의 일원으로 합류하게 된다면, 이러한 경험과 기술로 프로젝트의 성공에 기여하고, 팀원들과 협업하여 시너지를 창출하고 싶습니다. 혁신적이고 도전적인 환경에서 제 역량을 발휘하며 함께 성장해 나가길 희망합니다.

업무 경험

PM
7월 2024 - 현재

<devhub>웹에서 ai 코드리뷰와 형상관리를 해보자

GIT에서 형상관리가 어려운 초보 개발자들을 위한 웹에서 쉽게할수 있는 스프링부트과 라마AI를 기반으로한 코드리뷰와 형상관리 사이트 입니다.

PM
5월 2024 - 6월 2024

개발자 커뮤니티 **DOGFOOT**

자바 스프링을 이용한 넷플릭스 형식의 개발자 정보 공유 커뮤니티 사이트 입니다.

학력

SQLD
2024

SQLD 개발자 자격증 취득

정보처리가사
2024

필기 합격 ~ 실기 진행중

KOSMO
2024

빅데이터 기반 인공지능 융합서비스 개발자 양성 과정 수료

건국대학교
서울
2018

학위 (최종학력)

- 영어학과, 경영학과

미시시피 주립대학교
미시시피
2017

학위 (최종학력)

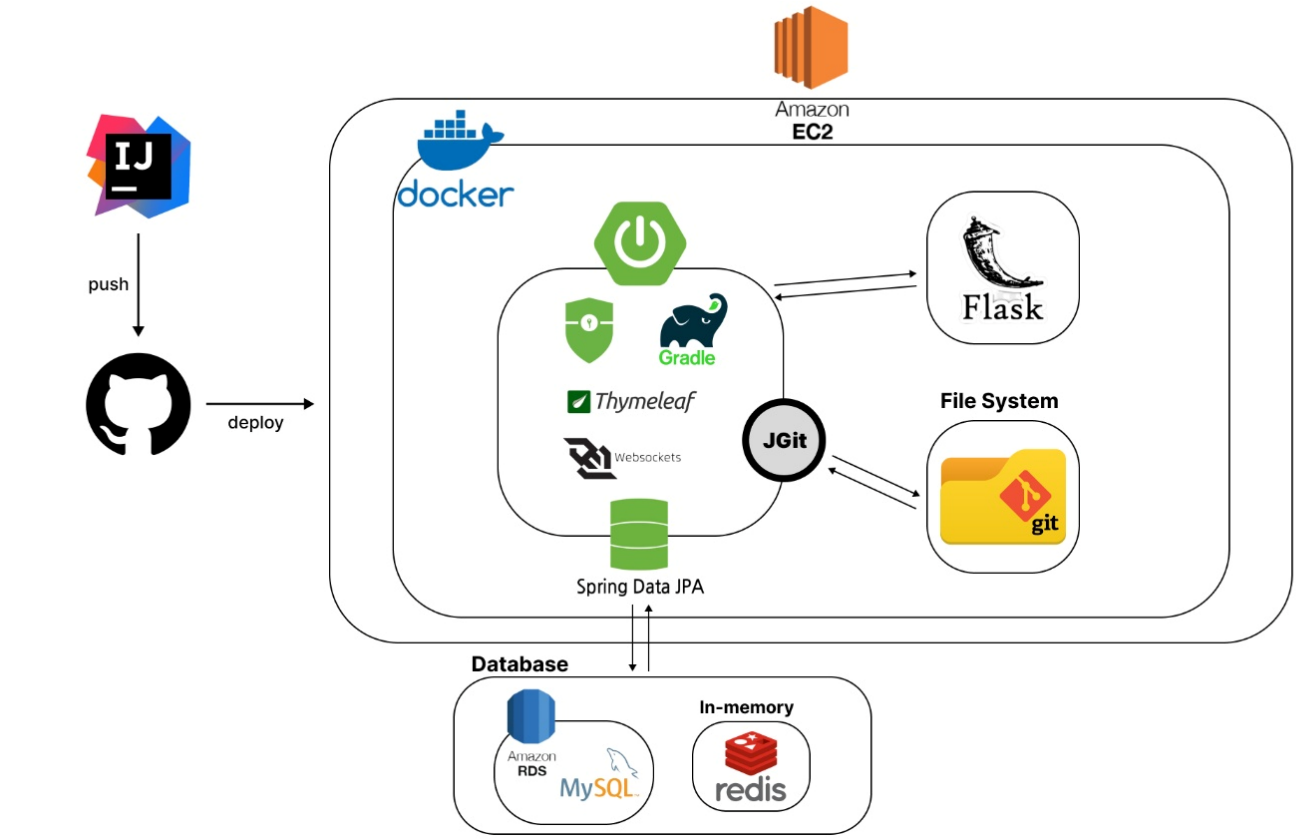
- 경영학과

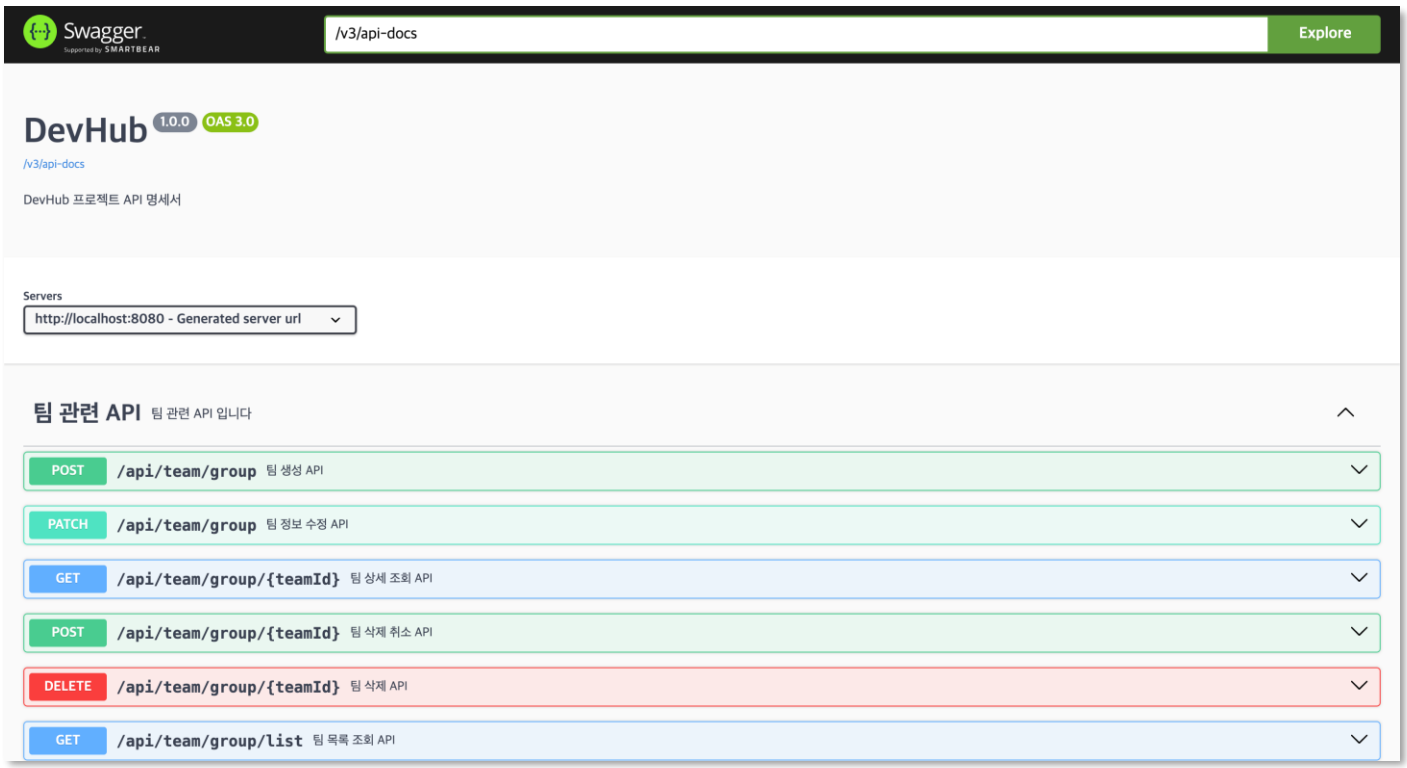
대표프로젝트 → Devhub : 깃보다 쉽게 '형상관리'와 '코드리뷰'를 해보자.

2024년 7월 24월 (5명 PM ~~ 2명 진행 중)

Api 문서화 내용 총 **43**개 (Swagger를 이용한 문서화)

아키텍처 및 API 문서화





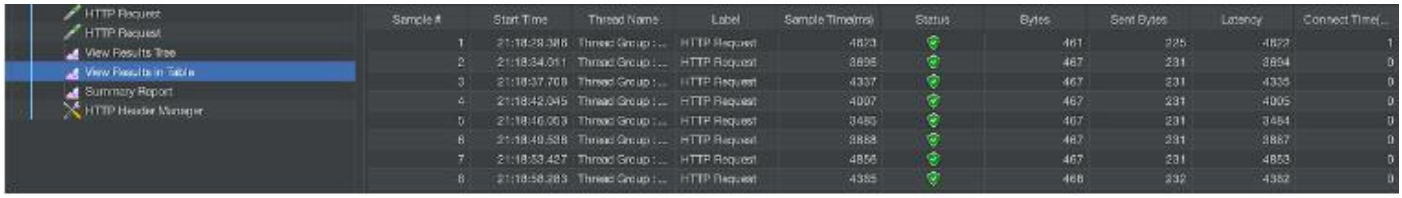
<프로젝트 진행 중 어려웠던 사항들과 해결방안>

(1) 이메일 관련 Api 전송속도 문제

JMeter 테스트 도구로 이메일 인증 기능을 테스트하는 과정에서 요청에 대한 응답 시간이 오래 걸리는 결과를 얻었을 때였습니다.

[테스트] (테스트 도구 : JMeter, Gmail : 1개 , Naver mail : 2개 , Temp mail : 6개)

1. 8개의 각 요청에 대해 쓰레드를 1개씩 생성하고 1초 동안 한 번만 요청을 하도록 설정합니다.



결과는8개의 요청은 모두 성공했습니다. 하지만 각 요청이 응답을 받기까지 평균적으로 **4.159초(4159ms)**가 소요되었습니다.

[문제 원인]

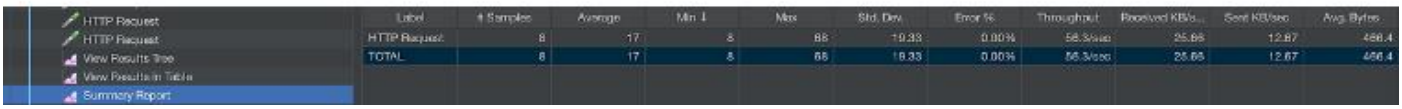
각 요청이 평균 4초의 응답 시간을 가졌지만, 8개의 요청을 모두 처리하는 데 **총 33**초가 소요되었습니다. 이는 JavaMailSender의 send() 메서드가 동기 방식으로 작동하기 때문에 발생한 문제입니다. send() 메서드는 SMTP 서버와의 통신 중 쓰레드를 블로킹하므로, 하나의 요청이 완료된 후에야 다음 요청이 처리됩니다. 이로 인해 모든 요청이 순차적으로 처리되며 총 시간이 길어졌습니다.

[개선 방법]

이 문제를 개선하기 위해서는 동기 방식으로 처리되는 **JavaMailSender를 비동기 방식**으로 처리되도록 하는 것입니다. 이때, @Async 어노테이션을 사용하기 위해서는 다음의 조건을 충족해야합니다.

1. @EnableAsync 어노테이션을 명시하여 @Async가 붙은 메서드가 비동기 처리 되도록 한다.
2. @Async는 proxy를 기반으로 동작하는데 self-invocation 즉, 내부 호출에 의해 실행될 경우 proxy의 어드바이스에 의한 추가 작업이 수행되지 않기 때문에 내부 호출에 의해 실행되면 안된다.

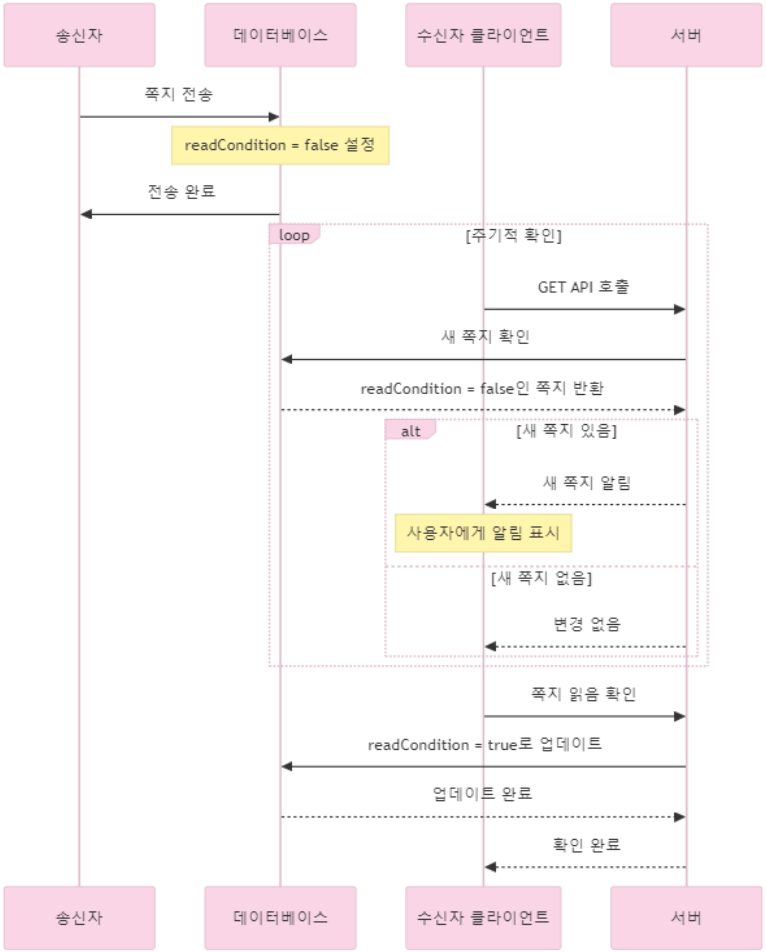
[개선 후 테스트 결과]



각 요청이 응답을 받는데까지 **평균 0.017초(17ms)**가 소요되었습니다.

(2) 쪽지 알림 관련

(2-1) 쪽지 시스템 다이어그램



-> **개념** : 쪽지가 도착했다는 것을 클라이언트에서 확인 가능하게 하기위해 DB에 readCondition으로 boolean 타입으로 만들어서 메시지가 도착해서 해당 값이 1이되면 서버로 전송

-> **기능** : **실시간 알림** 구현

클라이언트는 주기적으로 서버에 GET API를 호출하여 새로운 쪽지의 존재를 확인합니다.

이 방식은 웹소켓 구현에 비해 서버 리소스를 덜 사용하면서도 준실시간 알림을 가능하게 합니다.

-> **상태 기반 알림 처리**(* 자바스크립트에서 인터벌 함수를 줘서 주기적으로 **자동조회**)

쪽지 도착 시 'readCondition'을 false로 설정하고, 사용자가 읽으면 true로 업데이트합니다.

이를 통해 정확한 읽음 상태 추적과 알림 관리가 가능해집니다.

-> 사용자 중심 설계

사용자 인터페이스에 새 쪽지 알림을 즉시 표시하여 사용자 경험을 개선합니다. 읽음 확인 후 자동으로 알림이 사라지도록 하여 사용자의 불필요한 조작을 줄였습니다

(3) AI를 이용한 코드리뷰 시스템 구축

LLM 모델로 로컬에서 라마3.1 70B 모델을 사용하였으나 AI 코드리뷰 결과까지 **노트북에 사양이 못버텨서 매번 15~20**초정도 소모와 함께 렉이 발생해서 **진행 불가**.

*다른 백그라운드 프로그램을 사용을 제외한 권장사양

권장사양	라마	현재 노트북
운영체제	Windows 10/11, macOS Big Sur 이상, 또는 최신 Ubuntu LTS 버전	Windows11
프로세서	Intel i7 또는 더 높은 성능의 AMD Ryzen 프로세서	Intel core ultra7 Gpu: Intel arc
메모리	16GB	16GB
하드 드라이브	SSD 필수, 최소 100GB 이상의 여유 공간	없음

-> **Open ai GPT** 로 코드 변경 (5초 안팎으로 리뷰결과 가능) → Api 사용 비용 발생

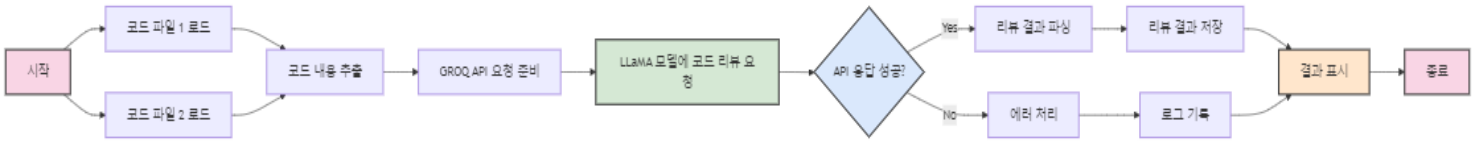
비용 문제를 최소화하기위해 최대 토큰 300으로 사이즈 조정하였으나 무료로 이용하고 API로 인한 네트워크 지연 시간이 없는 로컬에 설치한 **라마 3.1** 이용하기 위해 리팩토링 작업

-> **groq**를 이용한 라마 api 사용

-> 무료 요금

-> 토큰크기 자유롭게 조정가능

-> 빨라진 응답속도 보장 **(0.45초)**



시스템 주요 기능:

(1) 다중 파일 동시 분석: 두 개 이상의 코드 파일을 동시에 처리하여 비교 분석
(2) AI 기반 코드 리뷰: 코드 품질, 최적화 가능성, 보안 취약점 등을 자동으로 식별
(3) 상세한 피드백 생성: 개선 사항에 대한 구체적이고 실행 가능한 제안 제공

남은 기술적 도전과 해결:

(1) 대용량 언어 모델의 효율적 통합: 최적화된 API 호출 및 캐싱 전략 구현 -> 로컬 라마3.1로 전환 예정
(2) 비동기 처리의 복잡성 관리: 스프링의 비동기 기능을 활용한 견고한 에러 핸들링
(3) 보안 고려사항: API 키 관리 및 민감한 코드 정보 보호를 위한 보안 계층 구현 -> Nginx 프록시 우회 공부 중

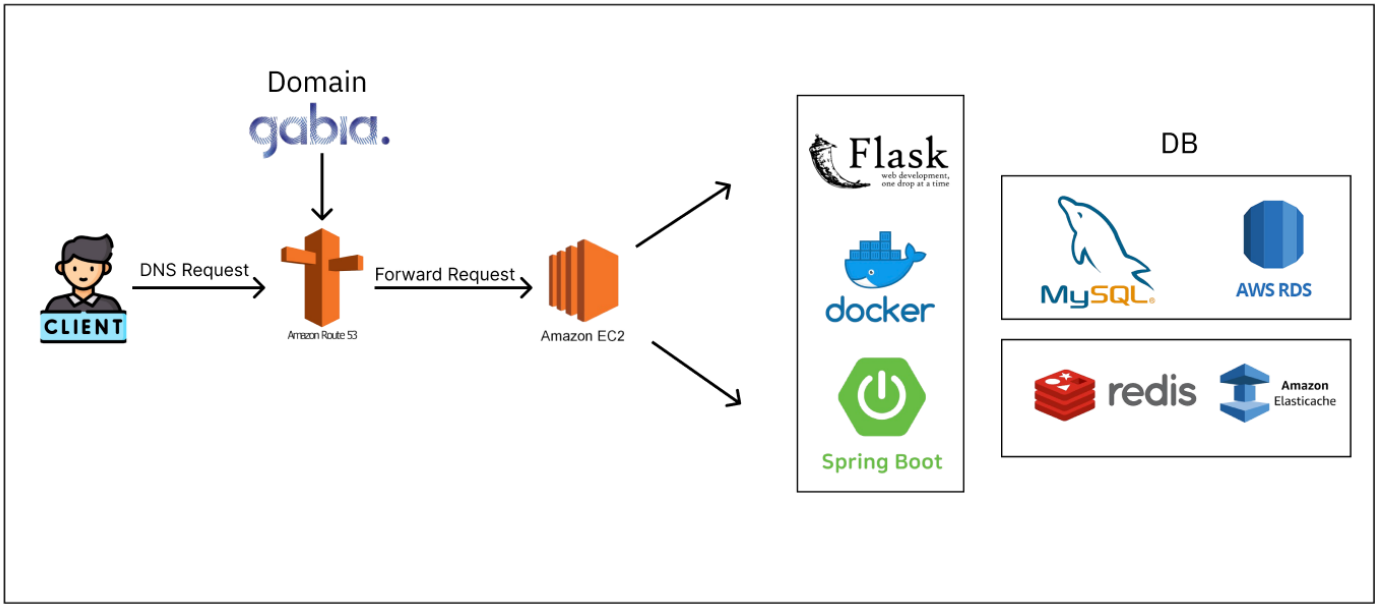
(3) 플라스크 프로젝트와 스프링부트 프로젝트 한 번에 도커와 AWS 이용해서 배포하기

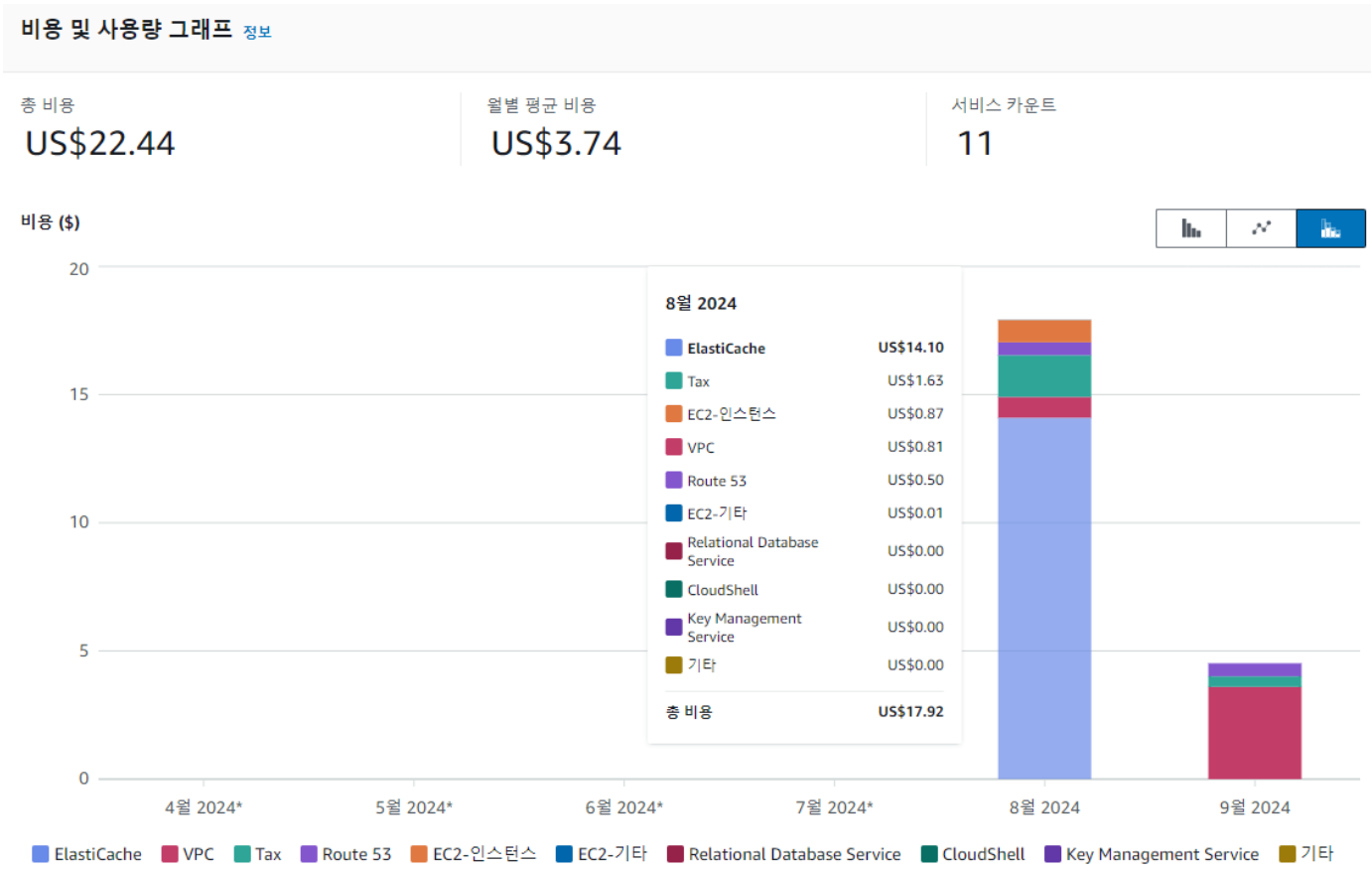
-> 플라스크와 스프링부트 각각 도커설정 파일 작성 후 도커 컴포즈 파일 생성

-> DB는 (MYSQL-> RDS , REDIS ElastiCache)

-> EC2(우분투, 프리티어 tc2 micro 사용) , 도메인 구매는 가비아에서 구매후 Route53으로 연결

(3-1) AWS를 통한 배포 아키텍처





➔ 실수 한 점 : Redis 연결하는 **Elasticache에서 노드 설정을 잘못해서** 프리티어지만 과금청구

➔ 현재 아직 계속 개발 중이기에 프리티어는 중지가 안되서 삭제하고 서비스 배포 준비를 위한 코딩 중