

# 차세대분산시스템 최종보고서

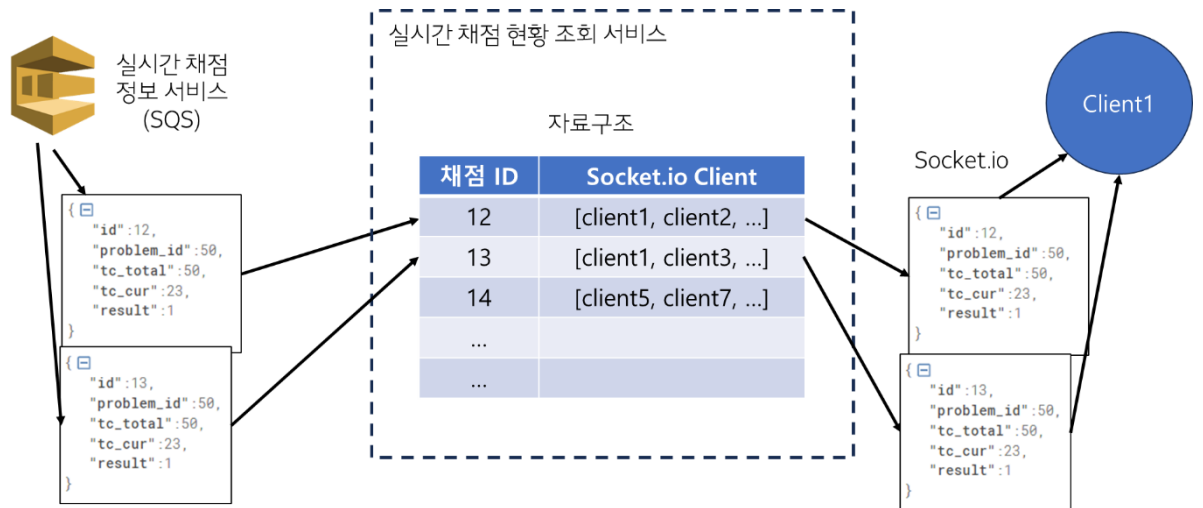
MSA 기반의 온라인 저지 시스템

Docker Online Judge

8팀 - 커여운 도커

## 목차

1. 구성원 소개 .....	4
2. 작품 소개 .....	5
2.1. 작품 연구 개발 동기 및 개념도 .....	5
3. 핵심 문제 .....	6
3.1. 핵심 문제 정의 및 해결 방안 .....	6
3.2. 관련 연구 개발 정리 및 참고자료 .....	7
3.3. 해결 방법 .....	10
a. 채점 환경의 확장성 .....	10
b. 채점 환경의 격리 및 신뢰성 보장 .....	11
c. 통신 동기화 .....	11
4. SW 설계 .....	11
4.1. 기능 요구 사항 .....	11
4.3. Use Case .....	12
4.4. Sequence Diagram .....	13
4.5. SW Architecture .....	14
4.6. 모듈/서비스별 상세 SW Architecture .....	14
4.6.1. 프론트엔드 .....	14
4.6.2. API Gateway, Service Discovery .....	15
4.6.3. Auth Service .....	15
4.6.4. Problem Manage Service .....	16
4.6.5. Submission Service .....	19
4.6.6. Judge Service .....	20
4.6.7. Message Queues .....	28
4.6.8. Real Time Service .....	29



.....	29
5. 최종산출물 형태.....	29
5.1 Deployment Diagram .....	29
5.2. Distributed Layered Architecture .....	30
5.3 실제 결과물 .....	31
6. Risk Analysis & Risk Reduction Plan .....	34
6.1. 완료한 Risk 목록 .....	34
6.2. 해결할 Risk 목록 .....	36
7. Success Criteria & Test.....	36
8. 최종 결과 및 분석 .....	38
9. 상세 스케줄과 상세 역할 분담.....	39
참고 자료 목록.....	40
API Gateway & Auth Service .....	40
Message Queue .....	40
Judge Service .....	40
Submission Service.....	41
Problem-Manage Service .....	41

## 1. 구성원 소개



백종원



김창엽



정우철

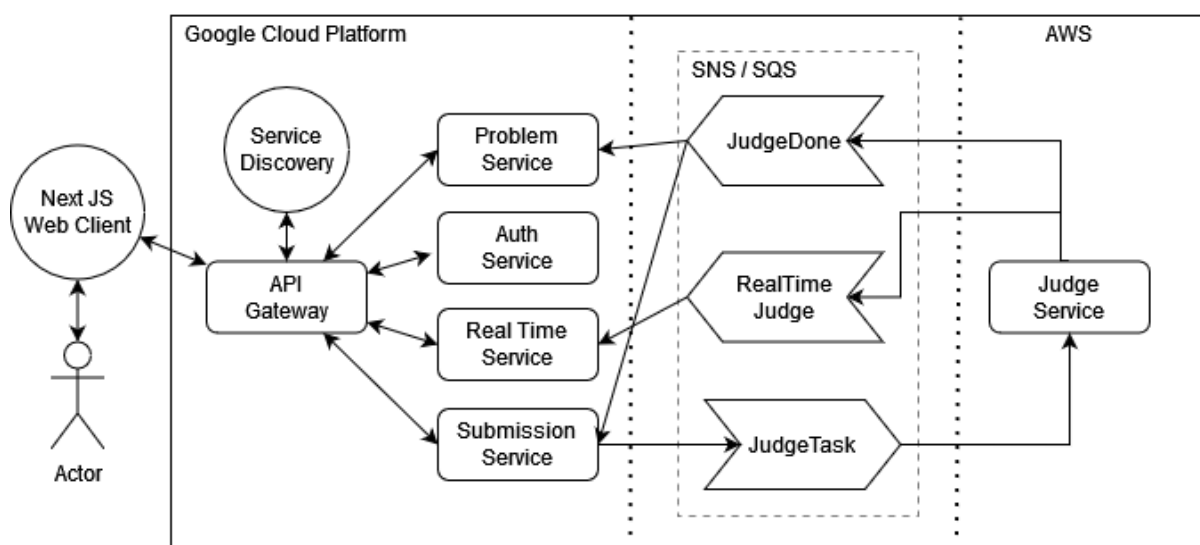


강병우

번호	이름	학번	역할	역할 상세
1	백종원	201811263	팀장, 백엔드	제출 서비스
2	김창엽	201811244	채점 엔진	격리된 채점 환경 구축
3	정우철	201811293	백엔드, 프론트엔드	API Gateway & Eureka 사용자 서비스, 실시간 채점 서비스 Frontend
4	강병우	201911146	백엔드	문제관리 서비스

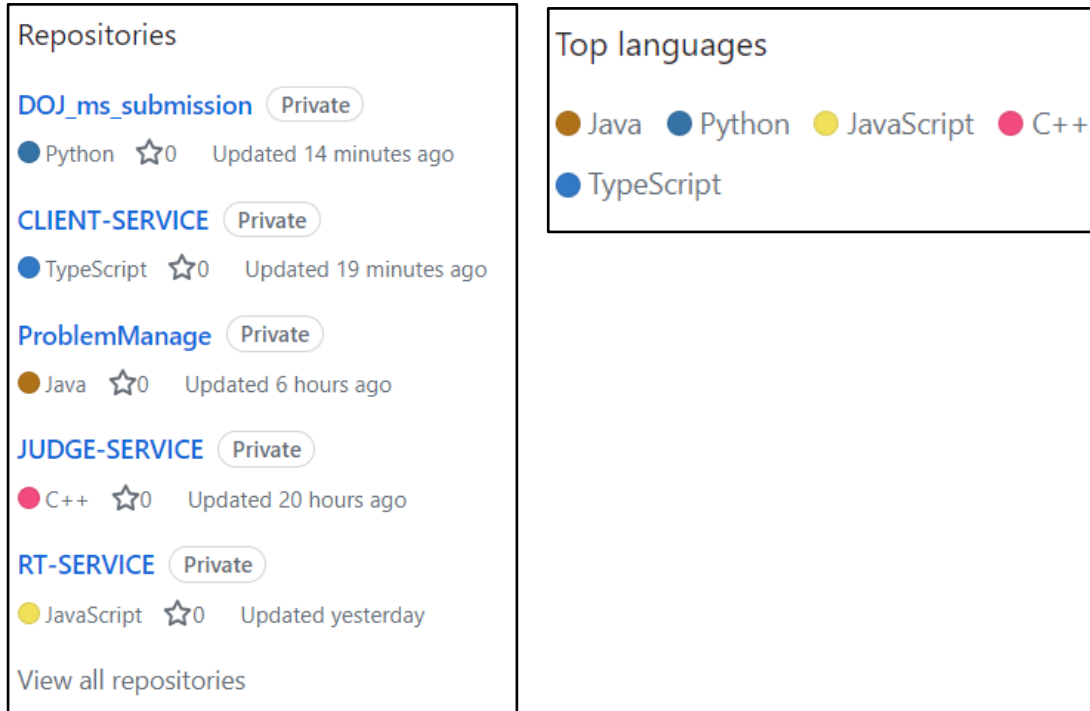
## 2. 작품 소개

### 2.1. 작품 연구 개발 동기 및 개념도



온라인 저지는 알고리즘을 구현하는 실력을 연습하기에도 좋고, 기업 입사시험으로 이

를 도입하는 추세이기 때문에 그 중요성이 점점 커지고 있다. 온라인 저지는 격리된 환경에서 사용자가 업로드한 소스를 실행하는 구조인데, 만약 모놀리식한 구조로 구성될 경우 신뢰성 있는 채점을 보장하면서 서버를 확장하는데 어려움이 존재한다. 따라서 유연한 Scale-Up 과 격리된 환경을 제공할 수 있는 MSA(마이크로 서비스 아키텍처)와 컨테이너 환경을 사용해서 이를 구현하였다.



실제로 개발 언어도 그렇고 다양한 환경에서 개발되어 MSA의 장점을 잘 활용하였다.

### 3. 핵심 문제

#### 3.1. 핵심 문제 정의 및 해결 방안

##### 1. 채점 환경의 확장성

- 채점 환경은 여러 언어를 확장하기 용이한 형태여야 한다.
- 급격한 제출 요청의 부하에 대응할 수 있도록 확장하기 편한 구조여야 한다.

##### 2. 신뢰성 보장

- 채점 서비스의 격리된 환경 제공

채점 환경은 격리된 환경에서 제출된 코드를 실행하여야 한다. 코드 실행 시 채점 서버에 영향을 미쳐선 안 된다.

- 신뢰성 보장

동일한 입력에 대해 항상 일정한 결과를 출력해야 한다. 동일한 제출된 코드에 대해 항상 일정한 채점 결과를 출력할 것이다.

### 3. 데이터 동기화

서비스 간 분산되어 있는 데이터에 대한 동기화 방법을 마련해야 한다. 특히 채점 서비스와 문제 관리 서비스 간의 데이터 동기화가 중요하다. 문제 관리 서비스에서 문제를 삭제했고, 채점 서비스에도 이 삭제가 반영되어야 하는데 삭제되어야 할 문제를 채점한다면, 신뢰성 있는 결과를 보장할 수 없다.

#### 3.2. 관련 연구 개발 정리 및 참고자료

프로젝트 구성 요소마다 개발에 필요한 기술들을 정리하면 다음과 같다

##### 1. API Gateway, Service Discovery

API Gateway는 여러 마이크로 서비스로 구성된 시스템에서 클라이언트와 서버 간의 통신을 관리하고 조정하는 역할을 한다. 이 시스템에서의 주요 역할은, 클라이언트의 요청을 적절한 마이크로 서비스로 라우팅하거나, 필요한 경우 서비스의 프록시 역할을 수행하여 클라이언트와 마이크로 서비스 간의 통신을 중개하는 것이다. 이용된 프레임워크는 Spring Cloud Gateway를 이용하였다.

Service Discovery는 동적으로 변화하는 서비스들의 위치와 특성을 찾고 관리하는 메커니즘을 나타낸다. 각 마이크로서비스는 자신이 어떤 기능을 제공하며 어디에서 실행 중 인지를 서비스 디스커버리에 등록한다. 서비스가 시작되면 디스커버리에 자동으로 등록되고, 종료되면 해당 서비스의 등록 정보가 해제된다. 이때, 등록 정보는 서비스의 IP주소와 port정보이고, 만약 하나의 서비스에 여러 개의 인스턴스가 등록된 경우, 이 인스턴스들 중에서

##### 2. Message Queue

서비스 간 통신에는 Message Queue를 사용한다.

메시지 큐를 사용하는 이유는 크게 다음과 같다.

###### 1. 비동기 메시징

어떤 이벤트에 대해 바로 처리하지 않고 이를 큐에 넣어두어 이후 처리를 담당하는 서비스들이 가능한 상황에서 가져가서 완료할 수 있도록 한다. 따라서 각 서비스는 메시지를 생성한 후 그에 대한 처리를 기다릴 필요 없이 자신의 로직으로 복귀할 수 있다.

###### 2. 배치 처리

메시지들을 큐에 넣어두고, 이를 필요로 하는 서비스에서 배치로 가져와 처리할 수 있도록

록 해 입출력, 통신 등에 낭비되는 시간을 줄일 수 있다.

### 3. 확장성

서비스 간 통신에 대한 부분을 메시지 큐에 위임하면서 메시지 처리와 실제 비즈니스 로직을 분리하는 것이 가능하다. 이로 인해 메시지 처리와 관련된 확장성이 필요할 시 정확히 해당 부분만 확장시키는 것이 가능하다.

이에 대한 명확한 구현체로서 AWS에서 제공하는 SQS와 SNS를 사용하기로 결정하였다.

SNS(Simple Notification Service)는 Publish/Subscribe 패턴의 메시징, 이벤트 기반 컴퓨팅을 지원한다. 이에 더해, FIFO(선입선출)을 제공한다. 비슷한 다른 Pub/Sub 메시징을 지원하는 도구들(e.g; Kafka)과의 차이점은 FIFO 보장이 있고, 이어서 설명할 SQS와 함께 쓰였을 때 더욱 안정적으로 동작한다.

SQS(Simple Queue Service)는 대기열 시스템으로 FIFO를 보장한다. 이와 유사한 기능을 하는 오픈소스로 AMQP 프로토콜의 RabbitMQ가 존재하나, 굳이 SQS를 선택한 이유는 다음과 같다.

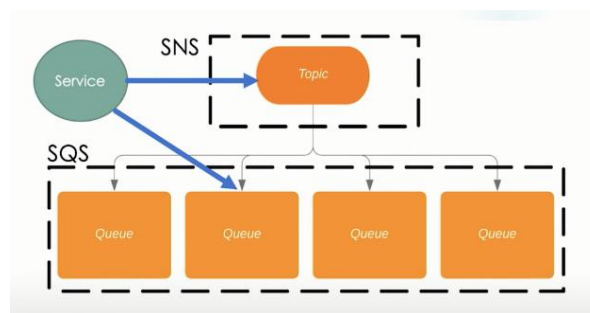


그림 1 -1 SQS/SNS Fanout Pattern Example

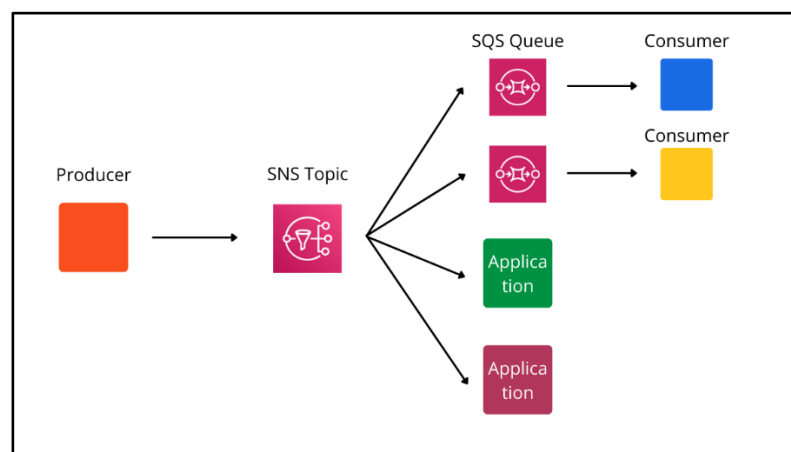


그림 2 - Essential Guide to SNS/SQS

SNS와 SQS를 혼용한 Pattern을 Fanout Pattern이라고 한다. 각 서비스는 자신이 발생시킬 이벤트를 한 Topic으로서 SNS에 발생시키고 실제 이를 처리해야 할 서비스들은 자신



의 SQS를 SNS를 구독하는 방식이다. 이를 통해 필요한 메시지들은 복사되고, Topic 생산자와 소비자를 분리시킬 수 있다.

### 3. 샌드박스 환경 제공 및 정확한 채점

온라인 저지는 결국 어떤 문제에 대한, 사용자의 풀이를 제출 받았을 때 이 풀이 (코드)를 컴파일하고 테스트 케이스들의 입력에 대해 알맞은 출력을 하는지 확인하여야 한다. 결국 코드를 “**실행**”하여야 하는데, 사용자의 코드를 온전히 신뢰할 수 있을까? 만약 제출된 코드에 `system(“sudo rm -rf /*”)`와 같은 악성 코드가 담겨 있다면? 서버를 다운시키려는 어떤 악의적인 제출도 충분히 있을 수 있기에, 보호된 영역 아래에서 실행시켜야 한다. 그래야 프로그램의 실행이 서버에 영향을 미치지 않게, 안전하게 실행할 수 있다.

이렇게 프로그램을 실행시킬 수 있는 보호된 공간을 샌드박스라고 하며, 믿을 수 없는 소스나 프로그램을 실행시킬 때 사용된다. 안드로이드에서도 커널 수준의 애플리케이션 샌드박스를 설정하여, 앱을 서로 분리하고 앱과 시스템을 악성 앱으로부터 보호한다.

온라인 저지에서도 서버를 신뢰할 수 없는 소스로부터 보호하여야 하기 때문에, 사용자가 제출한 코드를 어떤 격리된 공간에서 실행할 수 있어야 한다. 이를 위해 리눅스 커널의 **namespace**와 **cgroup**을 사용하여 격리된, 제한된 공간에서 사용자의 코드 실행이 가능하다.

IOI (The International Olympiad in Informatics)에서는 이런 샌드박스 환경 제공을 위해 [isolate](#)라는 애플리케이션을 사용한다. 이 애플리케이션에서는 격리된 환경에서 코드를 실행할 뿐만 아니라, 온라인 저지에서 문제를 풀 때 전혀 필요 없고, 실행되면 문제가 있을 수 있는 **시스템 콜**들을 막는다.

- 파일 접근 (open, read, write, ...)
- 메모리 동적 할당 (brk, mmap, mlock, ...)
- 프로세스/쓰레드 생성 (fork, clone, vfork)
- 시그널 발신 (kill, killpg, tkill...)
- Inter-process communication (shmget, msgget, mq\_open, ...)
- 그 외 소켓 통신, execve, sleep, flushing buffers to disk

위 기능들은 문제 풀이에 전혀 필요하지 않은 기능이므로, 애플리케이션 자체적으로 막아 놓았다. 또한 cgroup으로 제출된 코드가 실행될 때, 실행 가능한 최대 시간과 최대 메모리를 설정할 수 있다. 문제 제한과 동일한 환경을 제공함으로써, 더 정확한 채점을 가능케 한다.

### 3.3. 해결 방법

#### a. 채점 환경의 확장성

- 다양한 언어 지원에 용이한 구조

현 채점 서비스는 사용자의 제출 정보를 메시지 큐로부터 전달받아, 해당 정보의 소스 코드부분을 파일로 만든 후, 컴파일하고 실행하는 구조이다. 단, 프로그래밍 언어마다 사용하는 메모리와 실행 시간이 다 다르기 때문에, 언어 별 시간/메모리 보정, 컴파일 명령어, 실행 명령어, 파일 확장자만 알고 있다면 얼마든지 새로운 언어에 대한 지원이 가능한 구조로 설계하였다.

- 컨테이너화

Robust한 제출 요청 발생시, 단일 채점 서버로 모든 제출 요청을 처리에 어려움이 존재한다. 따라서 채점 서비스에 대한 컨테이너화를 진행, 확장성을 확보해 여러 대의 채점 서비스로 처리량을 늘려 대응 예정이다. 특히 채점 서버를 기능 별로 나누어 채점을 더 필요로 하면 채점을 맡는 컨테이너를 늘림으로써 확장성을 보장하였다.

- 채점 큐 구조 수정

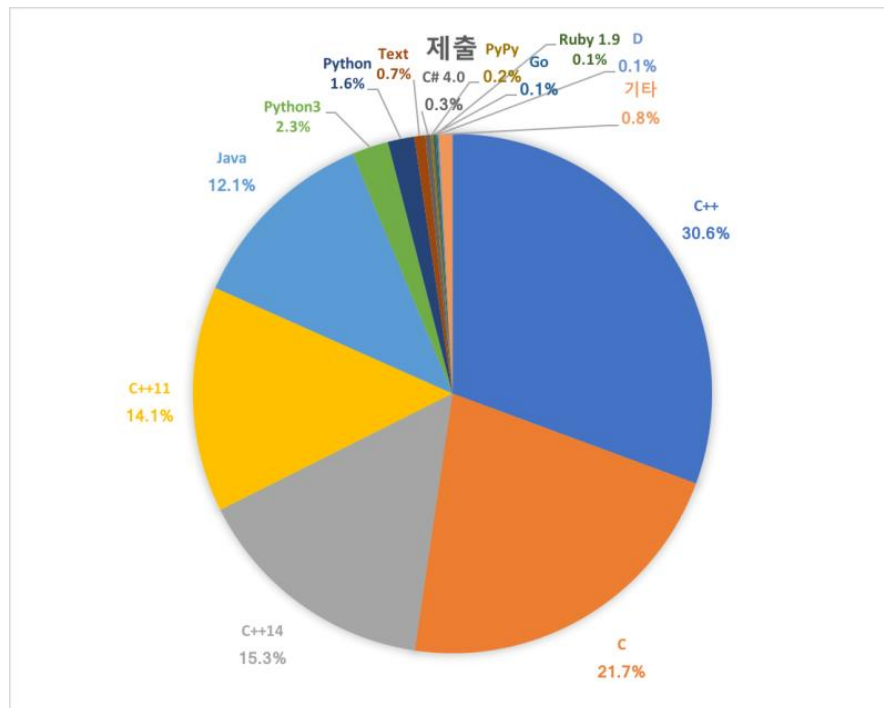


그림 2 - [2017년 3월 기준 백준 제출 통계](#)

C/C++ 계열의 언어들은 많이 제출되는 언어 중 하나이다. 물론 Pypy 포함 Python이나 Java의 제출도 적진 않지만, Problem Solving이나 Competitive Programming에서의 모든 문제들은 C++로 풀리게끔 보장이 되어 있고, 속도도 다른 언어들에 비해 빠르므로 많이

제출되는 언어이다. 따라서, C/C++로 된 제출 정보들을 담는 큐 (SQS)와 그 외의 프로그래밍 언어들을 담는 큐 (SQS) 두 개를 두어 여러 워커들로 하여금 위 큐들을 폴링하도록 하였다.

b. 채점 환경의 격리 및 신뢰성 보장

- 격리된 환경 제공:

ioi의 isolate라는 오픈소스 애플리케이션을 사용하여 제출된 사용자의 코드를 문제의 테스트 케이스마다 실행 중 격리된 환경을 제공한다.

- 신뢰성 보장:

채점 서버는 각 언어별 동일한 컴파일러로 컴파일 후 동일한 명령어로 실행할 것이다. 또한, 채점 서버에서 가용할 수 있는 메모리보다 문제에서 제시한 메모리가 더 적을 때만 채점을 하여, 항상 일관된 결과를 보장한다.

코드 실행 시의 Determinism을 보장해주기 위해, 호스트 머신의 커널과 시스템 설정을 조절해주었다. 대표적으로 page swap을 끄고, address space layer randomization을 막았다.

c. 통신 동기화

메시지 큐를 사용하면서 서비스 간 통신의 동기화를 확보할 수 있다. 모든 서비스는 타 서비스와의 통신 과정을 메시지 큐에 위임하면서 여러 이점들을 얻을 수 있는데, 통신 동기화 또한 이에 포함된다. SQS, SNS의 경우 모두 FIFO를 통해 메시지 순서를 유지할 수 있고, Duplication ID를 통한 중복 삭제가 가능하다. 또한 Fanout Pattern이 적용되어 있어, 동일한 이벤트 혹은 메시지를 이를 처리하려는 서비스 측에 그대로 전파시켜 줄 수 있다. 동시에 메시지를 발생시키는 서비스 측에서는 관련 사항을 이벤트 발생 이후의 사건들을 하나하나 추적, 관리할 필요가 없어진다는 장점 또한 존재한다.

## 4. SW 설계

### 4.1. 기능 요구 사항

1. 알고리즘 문제와 그에 대응하는 테스트 케이스가 존재, 이들에 대한 CRUD 기능 필요
2. 존재하는 문제 목록에서 문제 선택, 그에 대한 풀이로 소스코드를 제출 기능
3. 제출된 채점에 대해 실시간으로 채점 현황을 표현하는 기능 필요
4. 제출된 채점에 대해 소스, 에러 메시지, 걸린 시간 등 상세한 정보 확인 기능 필요

#### 4.2. 비기능 요구사항(성능, 신뢰성, 보안성, 안전성, 가용성)

1. 기본적인 API 호출은 최대 2초 안에 실행되어야 한다.
2. 제출의 컴파일 결과는 항상 같아야 한다.
3. 채점에 실패할 경우 예러 코드가 출력되어야 한다.
4. 메시지 토픽은 유실되어야 한다.
5. 도커를 지원하는 다양한 플랫폼에서 구동
6. AWS Infra 혹은 GCP Infra를 사용해야 한다.
7. 도커 컨테이너 환경으로 배포한다.
8. 서비스 구현에 있어서 언어 혹은 프레임워크는 자유롭게 선택, RDB 또한 자유
9. 메시지 큐를 사용하여 서비스 간 통신 및 데이터 동기화

#### 4.3. Use Case

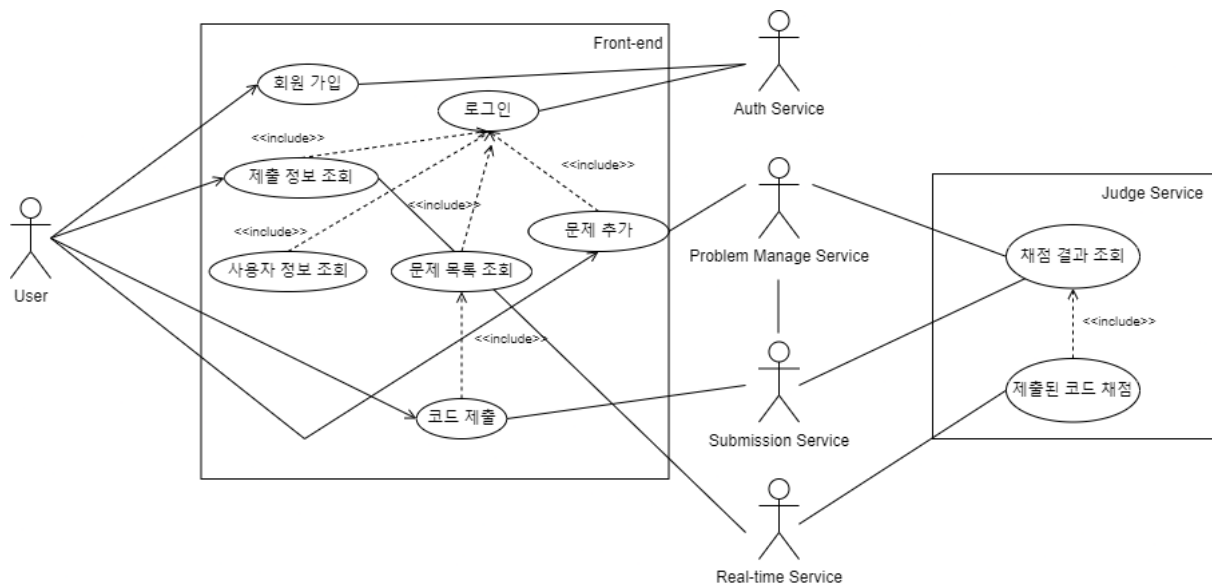


그림 3 - Use Case diagram

#### 4.4. Sequence Diagram

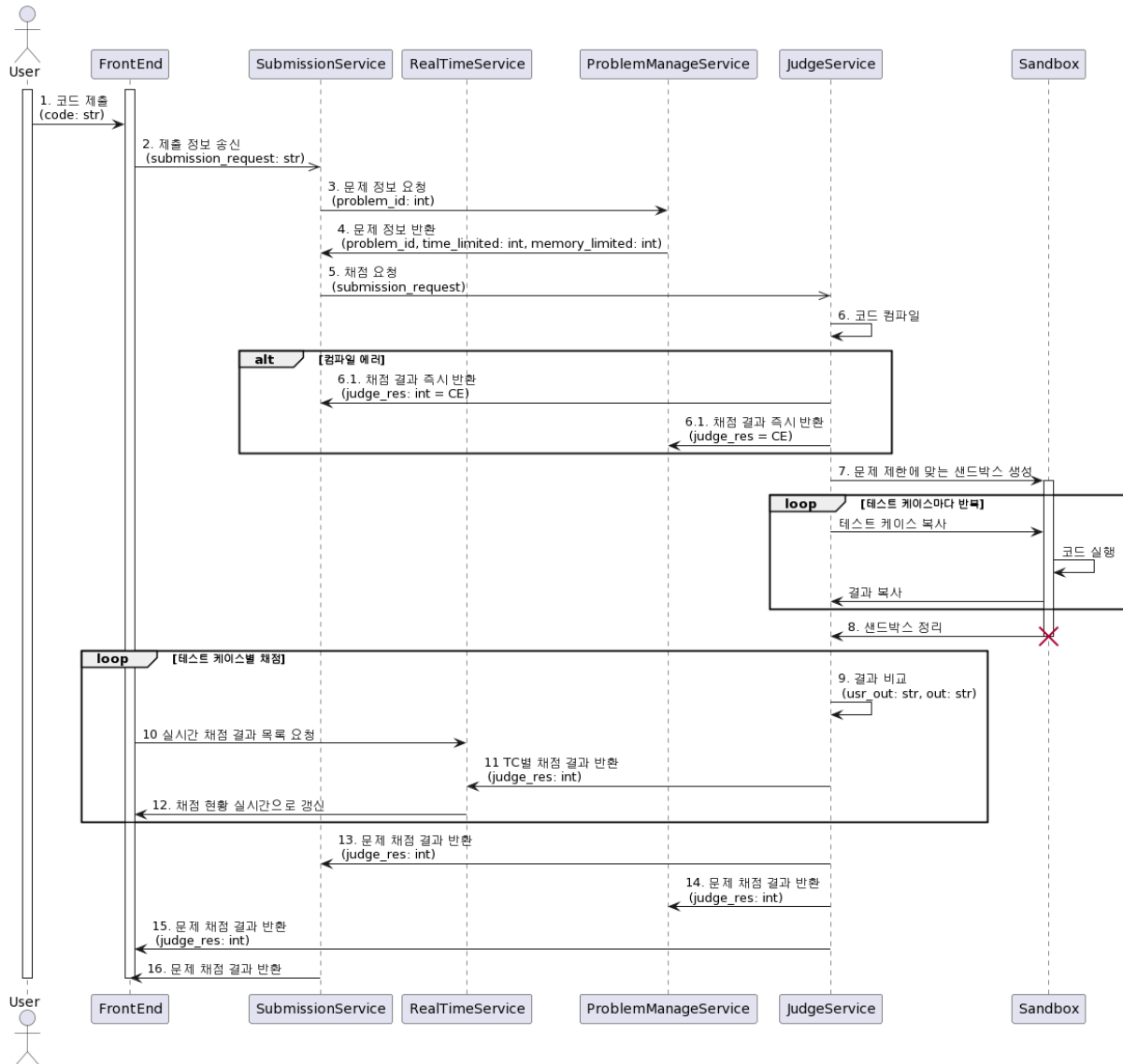


그림 4 – Sequence Diagram - Use Case: Submission

#### 4.5. SW Architecture

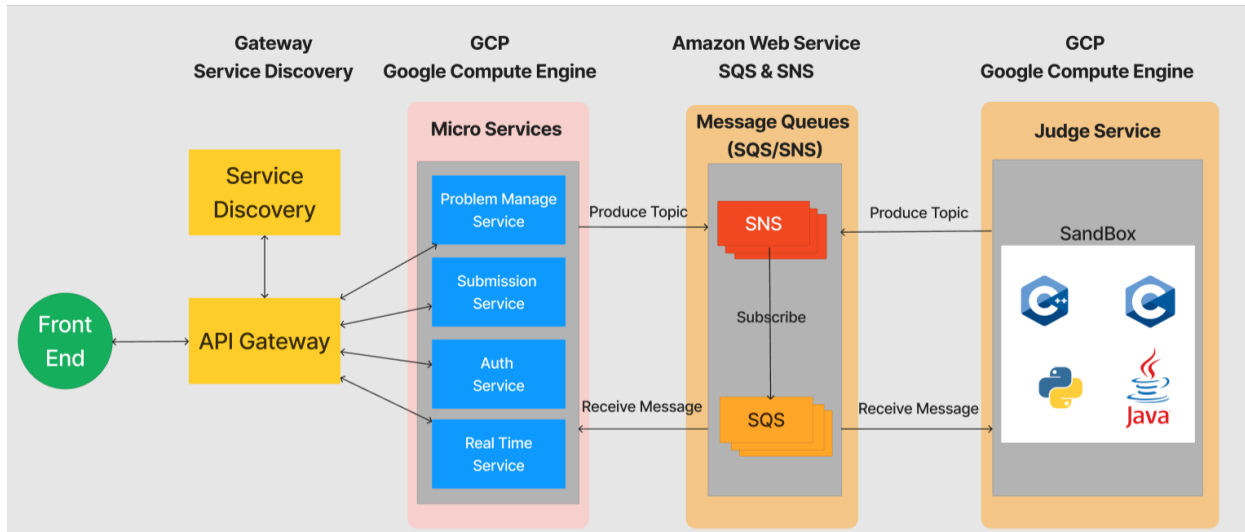


그림 5 - System Architecture

전체적인 아키텍처는 위의 그림과 같다. 사용자는 브라우저 상으로 제공되는 프론트엔드로 접근하며, 프론트엔드는 API Gateway를 통해 각 서비스들에 요청을 보낼 수 있으며, 채점 환경을 포함한 여러 서비스 간의 통신에는 SQS, SNS로 구현된 메시지 큐를 통해서 통신한다. 이때 프론트엔드는 Real Time Service를 제외하고는 HTTP로 통신하며, Real Time Service와는 WebSocket 을 통해 통신하여, 실시간 채점 현황을 획득하는 것이 가능하다.

#### 4.6. 모듈/서비스별 상세 SW Architecture

##### 4.6.1. 프론트엔드

프론트엔드는 최근 많은 이용률을 보여주는 React 기반의 프레임워크를 선택하였다. React는 CSR을 제공하는 라이브러리다.

대부분의 온라인 저지는 Google등의 검색엔진에 문제가 노출되기 때문에, 이를 보여주기 위해선 SSR이 권장된다. CSR의 문제는 대부분의 검색엔진 봇들이 제대로 된 메타 데이터를 가져가지 못하기 때문에, 이를 해결할 필요가 있는데, Next.JS라는 라이브러리는 React 기반으로 되어있으나, 검색엔진에 대응할 수 있는 SSR을 제공하기도 하기 때문에, 이를 이용하여 검색엔진에 노출되지 않는 문제를 해결할 수 있게 되었다.

현재 완성된 UI/기능 과 예정에 있는 UI/기능은 다음과 같다.

UI/기능	구현 여부
로그인	완료
회원가입	완료
문제(목록) 조회	완료
문제 추가	완료

문제 제출	완료
문제 제출(목록) 조회	완료
이용자 정보 조회	완료
실시간 제출 현황 조회	완료

현재 완성된 UI/기능 과 예정에 있는 UI/기능은 다음과 같다.

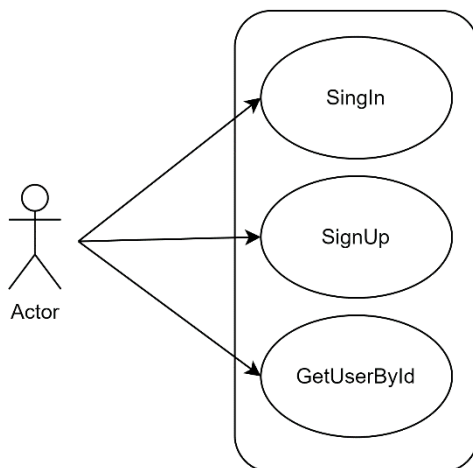
특히, 문제에 대한 답을 제출하거나 채점 정보를 조회할 때, 코드를 입력하거나 보게 되는데, MS에서 제공하는 VSCode 스타일의 Monaco Editor를 제공해 주어, 코드를 보는 가독성이 좋게 구성하였다.

이용자 정보를 조회하는 UI는, 이용자 정보와 전체 제출 회수, 성공/실패 제출 회수, 성공/실패한 문제 목록을 제공할 예정이다.

실시간 제출 현황 조회는 메시지 큐를 이용하는 Real Time Service를 새로 생성하여 이번 최종 발표에서 제공할 수 있도록 하였다. Node.js의 Socket.io를 이용하여 클라이언트와 통신할 수 있다. 채점 현황 목록을 조회하면, 채점이 완료되지 않은 요소를 Real Time Service에 등록하고, 실시간으로 등록된 채점에 대한 TC정보를 받아온다.

#### 4.6.2. API Gateway, Service Discovery

API Gateway는 여러 마이크로 서비스 아키텍처로 구성된 시스템에서 클라이언트와 서버 간의 통신을 관리하고 조정하는 역할을 한다.



가장 큰 역할은, 서버로 들어오는 요청들에 대해, 라우팅 및 프록시를 제공해주어 적절한 요청 경로에 해당하는 서비스로 연결해 준다. 또한, 일부 요청은 사용자 인증이 필요한 경로가 있는데, 이때, JWT토큰 인증을 검사하기도 한다.

API Gateway에서 연결할 서비스는 많이 존재하는데, 여러 서비스가 추가될 때 마다 동적으로 이를 반영할 수 있도록 하기 위해선 Service Discovery 도구가 필요하다. 이를 위하여 Eureka를 도입하여 각 마이크로 서비스가 실행될 때 마다 Eureka에 자기 자신을 등록시키도록 하였다. Eureka는 기본 설정이 서비스가 스스로 등록하는 Hostname을 기준 주소로 등록하도록 되어있으나 Hostname이 현재는 의미가 없는 수준이기에 접속 시 연결되는 IP주소로 연결되도록 하였다.

#### 4.6.3. Auth Service

Auth Service는 Python의 FastAPI를 기반으로 개발하였다. 기본적인 로그인, 회원가입, 이용자 정보 제공 API를 제공하고 있으며, JWT를 발급해준다.

이에 대한 API 목록은 다음과 같다.

기능	URI
로그인	/signin
회원가입	/signup
이용자 정보 제공	/user/{id}

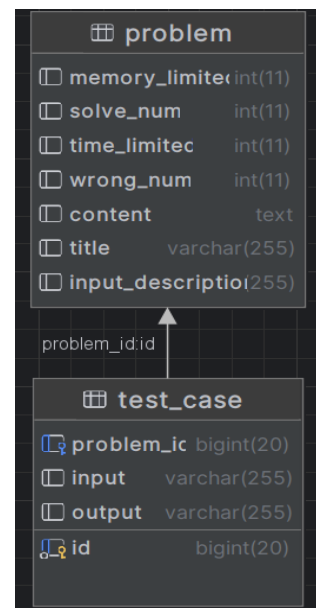
#### 4.6.4. Problem Manage Service

##### - 역할:

- 문제 및 테스트케이스 데이터 추가/수정/읽기/삭제 기능
- 채점 서버와 테스트 케이스 동기화 기능
- 채점 결과에 따른 횟수 카운팅 기능

##### - 주요 기술

백엔드 서비스는 Spring boot를 선택했다. 쉽고 빠른 개발을 위해 많은 생태계(JPA 등)를 지원하기 때문이다. DB는 Mariadb를 선택했다. 문제와 테스트케이스 간 관계를 나타내기 위해 RDB를 선택했다. 백엔드 관리하는 데이터는 크게 두 가지로 나뉜다. 문제를 관리하거나 테스트케이스를 관리한다. 왼쪽 ERD가 문제 관리 서비스에서 관리하는 데이터베이스 클래스이다. 또한 Fanout Pattern을 적용하여 SNS/SQS를 통해 Event-Driven Architecture를 적용하였다.



##### - 기능 설명

- API 서비스 : Problem/TestCase에 대한 CRUD 기능을 제공한다.

그림 76 - 문제 관리 서비스 테이블

#### Problem Service

API URL	Method	Description
/problem	post	문제와 테스트케이스 추가
/problem/{id}	get	해당 id의 문제 정보 조회
/problem/all	get	모든 문제의 제목과 설명 조회
/problem/{id}	delete	해당 id의 문제 삭제



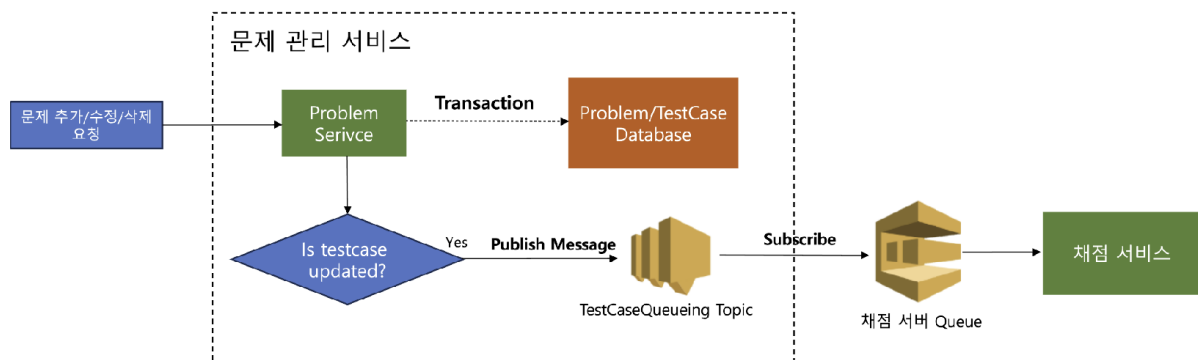
/problem/{id}	put	해당 id의 문제 정보 수정(테스트케이스 수정X)
---------------	-----	-----------------------------

### TestCaseService

API URL	Method	Description
/testCase	post	특정 문제의 테스트 케이스 추가
/testCase/{id}	delete	해당 id의 테스트 케이스 삭제
/testCase/{id}	put	해당 id의 테스트케이스 수정

#### ○ 문제 추가/삭제/수정 과정

- 문제 추가/삭제/수정 요청을 서비스에서 수신하면 로직을 거친 뒤, 수정사항을 DB에 반영한다.
- 이 때, TestCase에 변경점이 발생한다면 이를 TestCaseQueueing이라는 토픽에 메시지를 발행한다.
- 채점 서버가 폴링 중인 MessageQueue에서 이를 수신하여 테스트 케이스를 동기화시킨다.



- 발행되는 메시지의 key-value는 다음과 같다.

problemId : 1:N으로 연결된 문제의 아이디

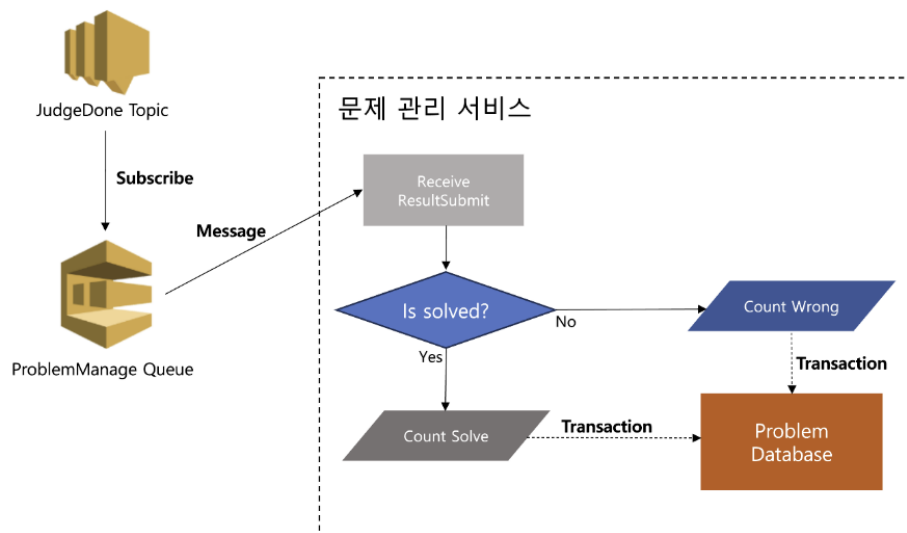
testCases : 동기화하고자 하는 테스트 케이스 리스트

EventType : ADD, UPDATE, DELETE 이벤트를 구분

Ex) 테스트 케이스를 추가할 때의 메시지 Payload

```
// ADD
{
  "problemId": 1,
  "testCases" : [
    {
      "id": 1,
      "input": "2 2\n2 2",
      "output": "1 1 1\n1 1"
    }
  ],
  "eventType": "TestCase_ADD"
}
```

- 채점 결과 수신 과정



- JudgeDone이라는 토픽을 구독 중인 ProblemManage Queue에 서비스 내부에서 SqsListener를 통해 비동기식 폴링을 한다.
- 메시지가 수신될 때, 메시지 Payload를 ResultSubmit 클래스로 변환한다.

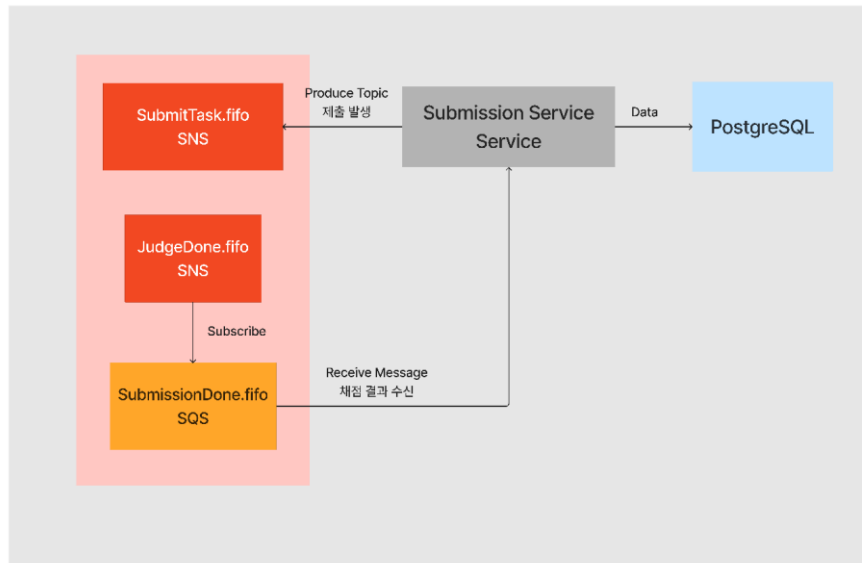
```

ap-northeast-2:26298138/273:00]-Judge-JudgeDone.Fifo:70ac67b5-d171-4100-8516-d1e
14710fd)
ResultSubmit(id=24, user_id=7, problem_id=18, language_code=1, memory_limited=100
time_limited=1000, judge_result=1, error_message=null)
2023-11-20T11:31:40.242Z INFO 1 [trap-executor-0] c.m.d.s.r.aws.ConfigCluster
  
```

judge\_result : 1-> Accepted

- 만약 채점 결과가 Accepted라면 맞춘 횟수를, 이 외에는 틀린 횟수를 카운팅하고 DB에 반영한다.

#### 4.6.5. Submission Service



Submission Service는 Python의 Django를 사용하여 개발하였으며, Database로 PostgreSQL을 사용하고 있다. 해당 서비스는 문제 번호, 입력한 소스, 언어 정보 등으로 정의되는 제출을 사용자로부터 전달받아 이를 생성한다.

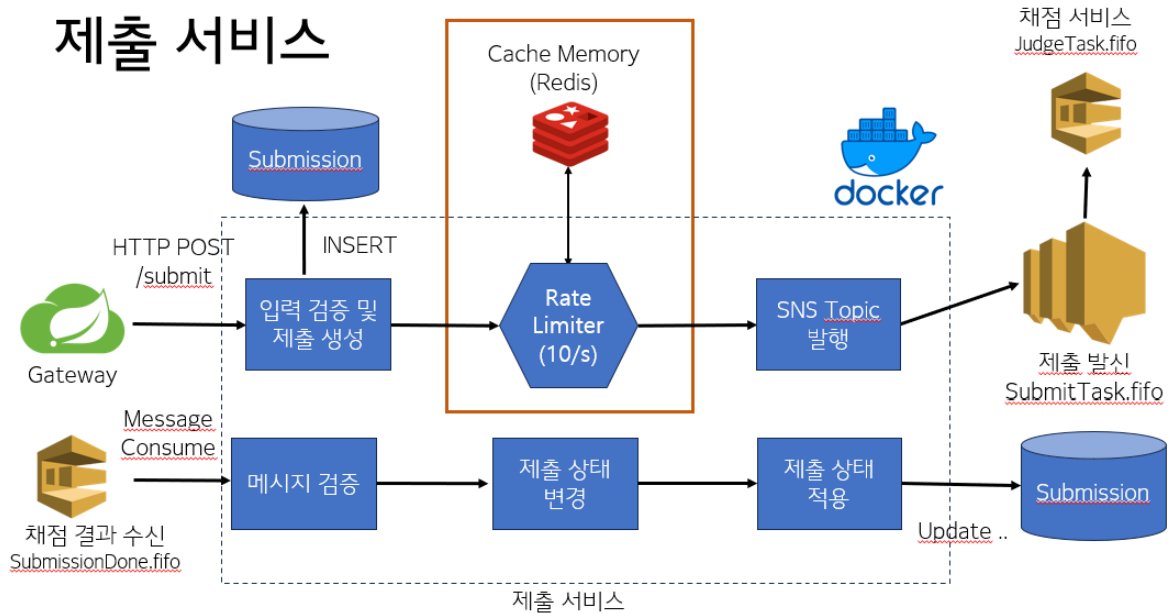
생성된 제출은 채점 환경으로 전달되어야 하는데, 프로젝트 전체에서 서비스 간 통신에 메시지 큐를 사용, 그 중에서도 SQS와 SNS를 사용하는 Fanout Pattern을 적용하기로 합의되었다. 따라서 해당 서비스에서도 그 규칙을 따라 제출 정보인 Submission을 생성, SNS "SubmitTask.fifo" Topic에 전달한다. 이후 이를 Subscribe 하는 SQS로부터 Judge Service 측이 가져갈 것을 기대한다.

이 제출로 인해 가능한 채점 서버로의 부하를 막기 위해 Submission Service는 채점 서버로 향하는 제출의 throughput을 제한할 필요가 있다. 이를 Throttling이라고 부르는데, Redis 와 같은 Cache Memory에 의존하여 구현하는 것이 일반적이다. 따라서, 일정 시간 동안 SNS Topic 생성을 제한하도록 Throttling을 구현하였다.

Judge Service 또한 마찬가지로 채점 완료 후 관련 데이터를 SNS "JudgeDone.fifo" Topic에 전달한다. 제출 서비스는 해당 SNS를 Subscribe 하는 SQS SubmissionDone.fifo에서 데이터를 수신하는 것이 가능하다.

수신된 채점 정보 메시지에서 Submission Service는 제출되었던 Submission의 상태를 자신의 Database에 업데이트한다.

이 모든 과정에 대한 그림은 다음과 같이 표현할 수 있다.



사용자는 자신의 풀이를 제출하는 것뿐만 아니라 제출했던 결과들에 대한 조회 또한 필요로 한다. 이에 따라 Submission Service는 제출 행위를 Database에 관리, 이에 대한 API를 제공한다. 그러한 API 목록은 다음과 같다.

API URL	Method	Description
/submit/{problem_id}	POST	문제에 대한 제출을 생성
/submission/{problem_id}	GET	문제에 대한 제출 내역 조회
/submissions?target={target}	GET	사용자의 모든 제출 이력 조회 (target=me) 모든 사용자의 모든 제출 이력 조회 (target=all)
/submit_detail/{submit_id}	GET	특정 제출 하나의 상세 내용을 확인
/user-submission-stats/{user_id}	GET	사용자의 맞은 문제, 틀린 문제 목록 조회

#### 4.6.6. Judge Service

- 역할:

- 제출된 코드를 **채점** 후 다른 서비스들에게 채점 결과를 **알린다**.
- 문제 관리 서비스와의 **동기화**
  - 채점 서버와 문제 관리 서비스가 갖고 있는 테스트 케이스들이 동기화 되어 있어야 한다. 따라서 문제 / 테스트케이스 CRUD 요청이 들어오면, 채점 서비스는 이에 맞게 처리를 해줄 수 있어야 한다.

- 기능:

채점 결과	값	설명
NJ	0	Not Judged. 채점하지 않은 경우. 채점 서버에 제출된 문제가 없을 때에는 NJ와 함께 에러 메시지로 해당 문제가 서버 내 존재하지 않는다고 알린다.
AC	1	Accepted. 모든 테스트케이스들에 대해 실행 결과가 일치하는 경우
WA	2	Wrong Answer. 테스트케이스 중 일치하지 않는 케이스가 있는 경우
CE	3	Compile Error. 제출된 코드 컴파일 중 에러 발생.
RE	4	Runtime Error. 코드 실행 중 런타임 에러 발생.
TLE	5	Time Limit Exceeded. 문제에서 제시한 시간 제한을 초과 시.
MLE	6	Memory Limit Exceeded. 문제에서 제시한 메모리 제한 초과 시.
OLE	7	Output Limit Exceeded. 출력 초과
SE	8	Sandbox Error. 샌드박스 환경 설정 및 실행간 에러 발생 시.

#### 채점 결과

제출 서비스에서 사용자의 제출 정보가 채점 큐를 통해 오면, 주어진 문제의 시간/메모리 제한에 맞게 샌드박스 환경에서 사용자의 코드를 실행하고 채점한다. 컴파일/코드 실행 도중 **에러**가 발생하거나 제시된 제한보다 더 많은 시간/메모리를 사용하였을 경우, 이를 채점 결과에 반영하여 다른 서비스들에게 결과를 publish 하여 알린다.

현재 채점 서버는 다음과 같이 다른 서비스들과 통신한다.

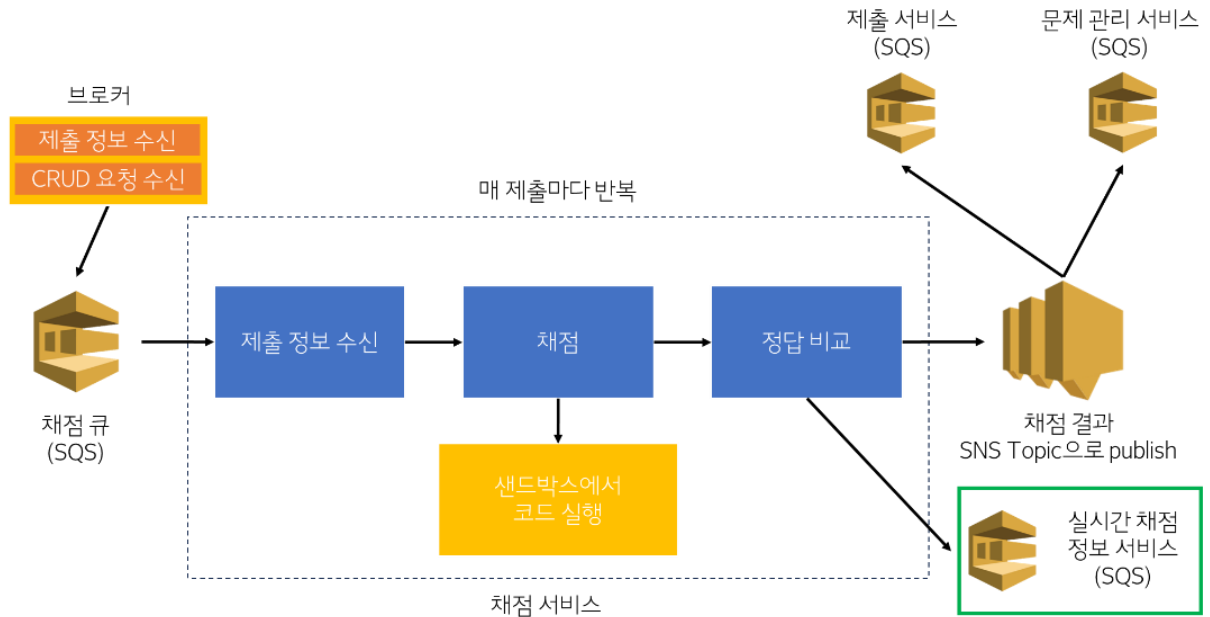


그림 87 - 다른 서비스와의 통신

- SQS: JudgeTask.fifo

#### 채점 서비스 <-> 제출 서비스

제출 서비스에서 사용자 제출 정보를 SNS topic으로 publish하면, 위 SQS queue는 해당 topic을 subscribe하고 있기에 queue 내에 제출 정보가 담기게 된다. 채점 서비스는 위 queue를 폴링하여, 들어온 제출 정보를 바탕으로 채점을 한다.

- SNS: JudgeDone.fifo

#### 채점 서비스 <-> 제출 서비스, 문제 관리 서비스

채점을 완료하여서 해당 제출에 대한 결과가 정해졌을 때, 채점 서비스는 그 결과를 [rapidjson](#)을 이용하여 JSON String으로 만든 후, 이 String을 SNS Topic으로 publish한다. 문제 관리 서비스와 제출 서비스가 각각 갖고 있는 SQS queue는 해당 topic을 subscribe하고 있어 SQS queue로 받을 수 있다.

- SQS: JudgeRT.fifo

#### 채점 서비스 <-> 실시간 채점 현황 서비스

한 문제에는 사용자의 코드 검증을 위한 여러 테스트 케이스들이 있는데, 매 테스트 케이스의 채점마다 채점 결과를 실시간 채점 큐로 보내, 프론트엔드에서 채점이 어디까지 진행되었는지 백분율(%)로 보여준다.

프론트엔드에서 문제 추가 요청을 하거나, 문제 관리 서비스에서 문제 / 테스트

케이스 관련 CRUD 요청을 보내면, 채점 서비스의 브로커가 해당 요청을 처리를 한다. 현재 구조 상 브로커와 워커가 한 호스트머신에 있고, 테스트 케이스를 바인드 마운트로 공유하고 있어 자연스럽게 워커들과도 동기화를 이룰 수 있다.

#### - Class Diagram

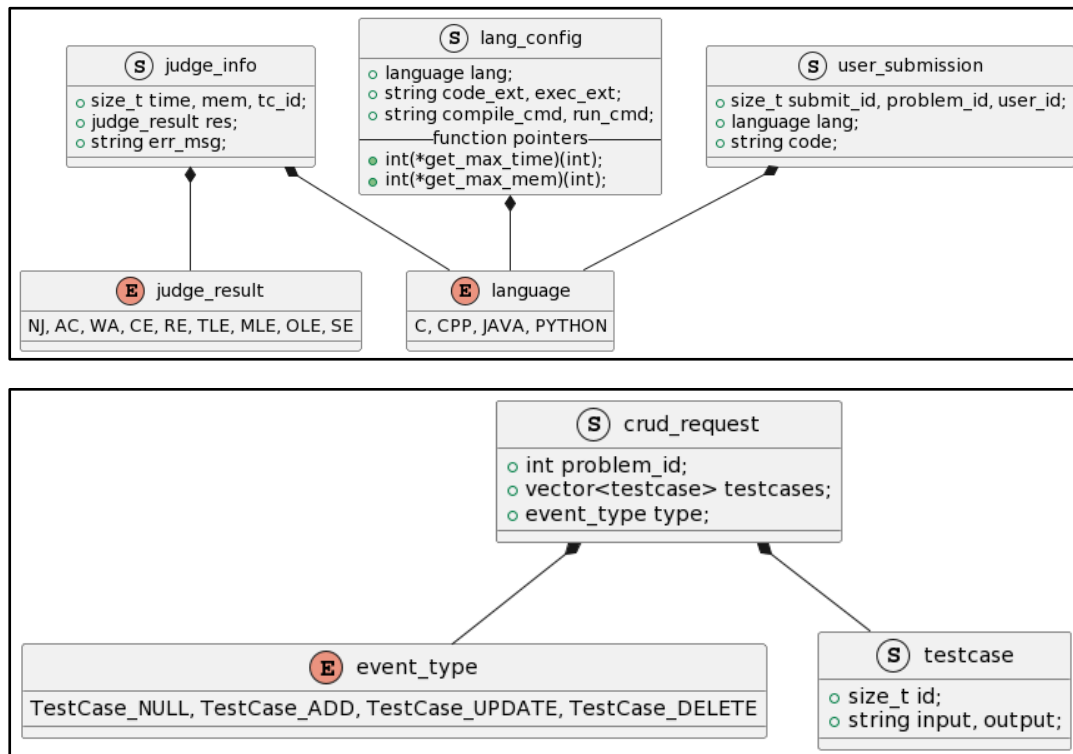


그림 98 - Class Diagram for Judge Service

언어 별 시간 복잡도/공간 복잡도 기준은 BOJ의 [기준](#)을 참고하여 작성하였다.

#### - 채점 서비스의 구조

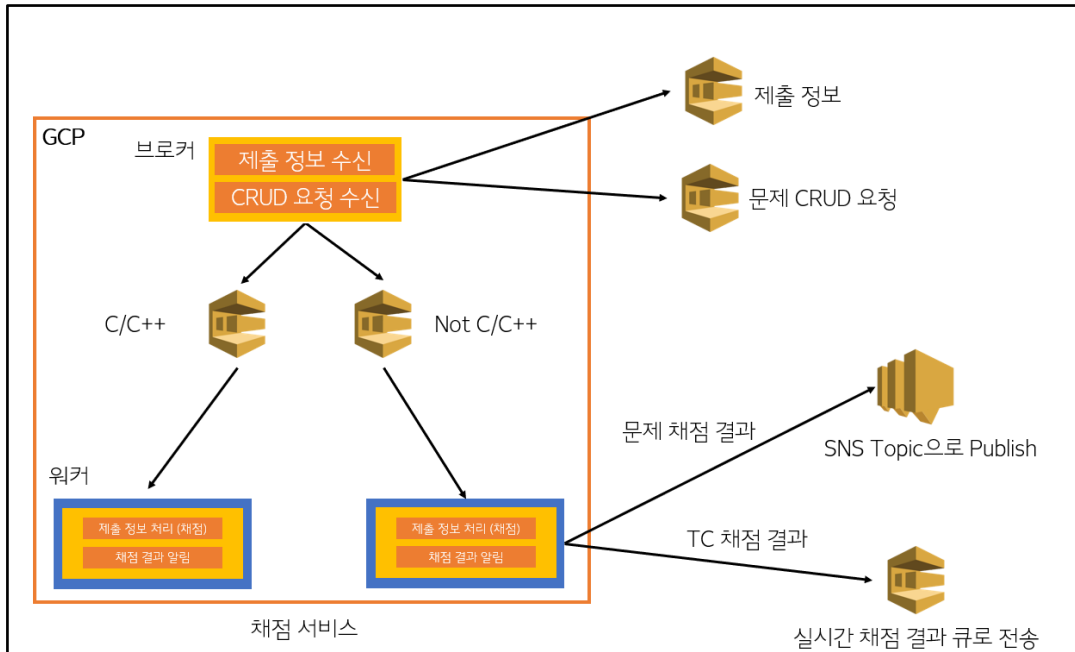


그림 9 - 채점 서비스의 간단한 구조

채점 서비스는 워커와 브로커로 나뉘어져 있다.

- 브로커: 문제/테스트케이스 CRUD 요청 처리, 제출 정보 전달
- 워커: 제출 정보를 채점

채점 큐 (SQS) 는 두 개의 큐로 나뉘어져 있다. C나 C++ 언어로 된 제출 정보들이 담기는 큐 와 그 외 다른 언어로 된 제출 정보들이 담기는 큐로 이루어져 있다. Problem Solving이나 Competitive Programming 환경에서는 C++이 빠르기도 하고, 모든 문제들이 C++로는 풀릴 수 있는 것이 보장되기 때문에 제출도 많은 편이다. 따라서 많을 수 있는 C/C++ 채점 요청들을 빠르게 해소시키기 위해 큐를 구분해 두었다.

이런 두 개의 큐들을 컨테이너화 된 여러 워커들이 풀링하고 있다. 한 개 이상의 워커들이 큐를 풀링하면서 문제들을 동시에 채점하고 그 결과를 SNS Topic에 발행할 것이다.

pridom1128/broker	1.5	Inactive	49 minutes ago	1.77 GB
	1.0	Inactive	17 hours ago	1.78 GB
pridom1128/worker	1.5	Inactive	9 hours ago	1.8 GB
	1.0	Inactive	17 hours ago	1.8 GB

현재 브로커와 워커 모두 Dockerhub에 올라와 있어, 언제든지 확장이 가능하다. 그리고 브로커와 워커는 현재 GCP 위에 배포를 다 마친 상태다. 이때 isolate는 cgroup v1만 지원하기 때문에, 클라우드 인스턴스의 cgroup 버전이 v2가 아닌 v1에서만 가능하여 OS의 버전을 Ubuntu 22.04 -> Ubuntu 20.04로 낮춰 클라우드에 배포하지 못하는 문제를 해결



하였다.

## - 채점 과정

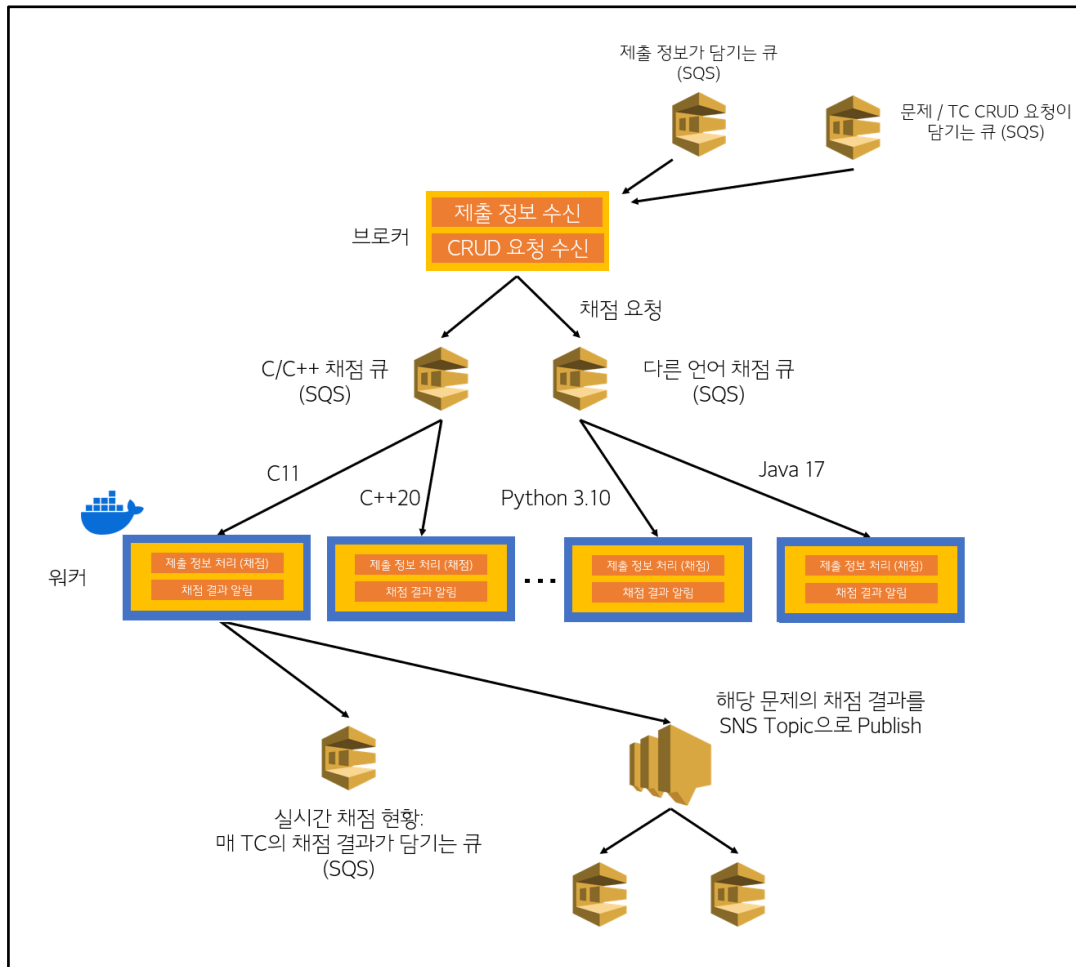
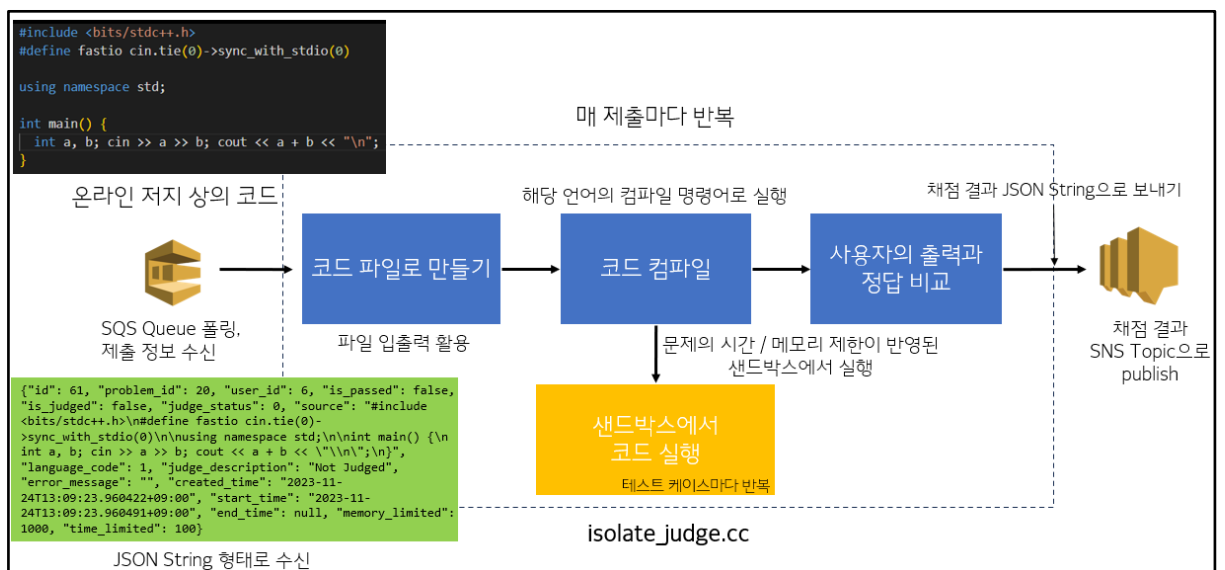


그림 10 - 채점 과정



- 1) SQS Queue에서 제출 정보 수신, rapidjson 이용하여 제출 정보 파싱 후 user\_submission 형태로 만든다.
- 2) 제출 정보의 code 부분을 파일 입출력을 이용, 파일로 저장한다.
- 3) 언어 설정에 저장된 컴파일 명령어로 해당 파일을 컴파일 한다. 이때 컴파일 에러 발생 시, 채점을 하지 않고 컴파일 에러라는 결과를 SNS Topic으로 publish한다.
- 4) 컴파일 후 애플리케이션과 테스트케이스의 입력들을 샌드박스 내로 복사한다.
- 5) 제출 정보의 문제 제한들을 이용, 애플리케이션을 샌드박스 내에서, 제한된 환경에서 실행한다.

```
std::string tc_num = i < 10 ? "0" + std::to_string(i) : std::to_string(i);
std::string cur_cmd = "isolate --cg --cg-mem=" + std::to_string(cur_config.get_max_mem(cur_sub.max_mem) * 1000)
+ " --time=" + std::to_string(cur_config.get_max_time(cur_sub.max_time) / 1000.0)
+ " --stdin=" + tc_num + ".in --stdout=usr_" + tc_num + ".out --run " + cur_config.run_cmd;
judge_info &cur_tc_judge_info = judge_res[i - 1];
```

주어진 메모리 제한과 시간 제한으로 isolate를 실행하여, 샌드박스 환경 내에서 컴파일 된 애플리케이션을 실행한다. 실행 도중 시간 초과/메모리 초과가 나면, 채점을 중단하고 결과를 보낸다. 이때, 매 채점 결과는 실시간 채점 결과로 보낸다.

- 6) 채점이 완료되면, 사용자의 출력과 테스트케이스의 출력 (정답)을 비교하여 AC인지 WA인지 판단하여 보낸다.
- 7) 채점이 끝난 후 샌드박스를 정리한다. 이 과정에서 생긴 불필요한 파일들을 전부 지운다.

**채점 결과**는 다음과 같이 생겼다.

```
{
  "id": 12485120,
  "user_id": 124125,
  "problem_id": 30461,
  "language_code": 1,           // CPP
  "memory_limited": 1024,      // MB
  "time_limited": 1000,        // ms
  "judge_result": 1,           // AC
  "error_message": null
}
```

그림 1011 - SNS Topic으로 보내는 채점 결과

language\_code와 judge\_result는 클래스 다이어그램의 language와 judge\_result와 동일한 값을 가진다. error\_message는 에러 발생 시, 당시 발생한 에러 메시지를 의미한다. 에러가 나지 않았으면

null로 보낸다.

```
=====
18 Statistics:
AC: 24
NOT AC: 0
Judge result: AC
Max Time: 276ms
Max Memory: 61108KB
Error Message:
=====
```

채점하는데 사용자의 코드를 실행하는데 걸리는 시간과 사용한 메모리의 용량 중 각각 최댓값들을 채점 결과에 포함한다. 만약 채점 중 에러가 났으면, 에러 발생 당시 에러 메시지도 채점 결과에 포함한다. 가령, `vector<int> arr(4)`에서 `arr[5]`와 같이 올바르지 않은 곳을 참조하였다면, Caught Fatal Signal 12와 함께 Segmentation Fault가 났다고 알려준다.

```
=====
18 Statistics:
AC: 4
NOT AC: 20
Judge result: RE
Max Time: 80ms
Max Memory: 5356KB
Error Message: exitcode: SIGSEGV
=====
```

### Submit Code

```
1  #include <bits/stdc++.h>
2  #define fastio cin.tie(0)->sync_with_stdio(0)
3
4  using namespace std;
5
6  int N, M, Q, W, P, ocean[200][200];
7
8  int main() {
9      fastio;
10     cin >> N >> M >> Q;
11
12     for(int i = 0; i < N; i++) {
13         for(int j = 0; j < M; j++) {
```

Submitted At: 2023. 12. 18. 오후 7:44:32

Program Language: C++20

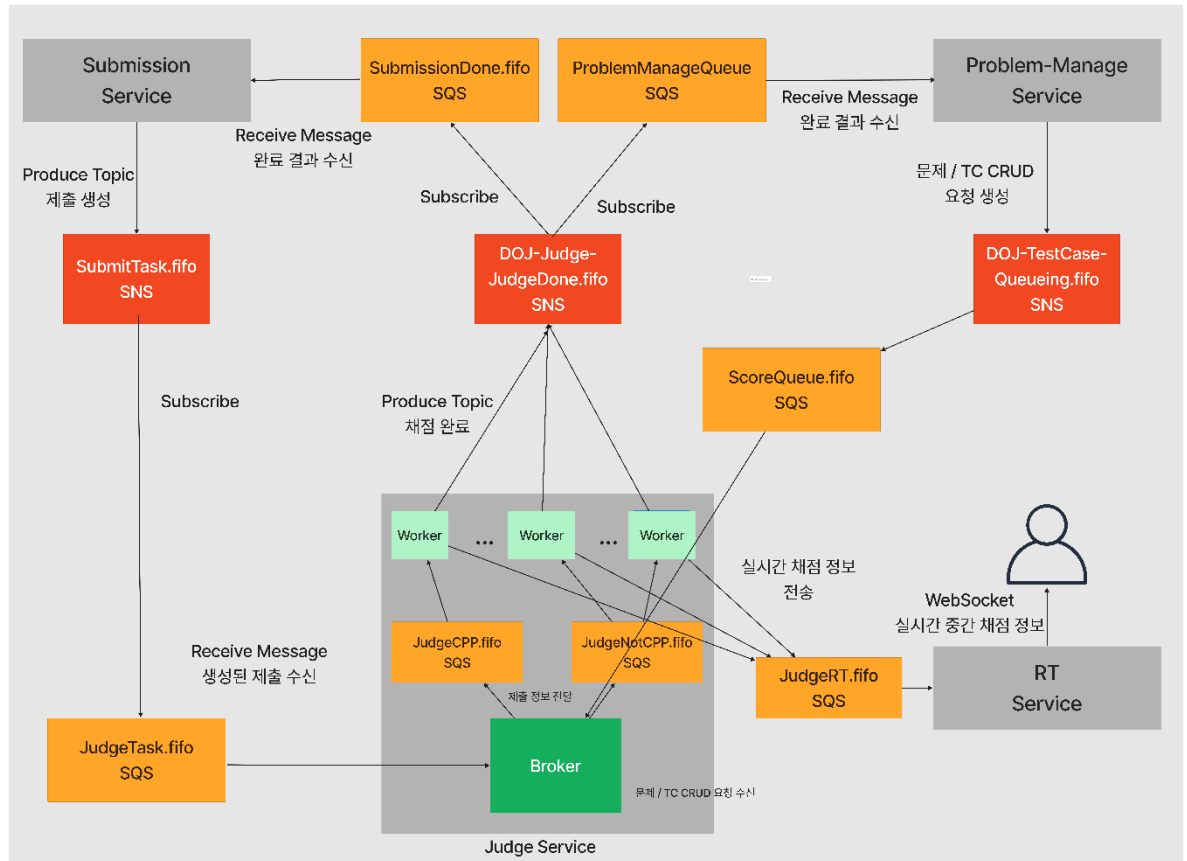
Result: Runtime Error

그림 12 - invalid reference 예시 (서버가 위, 프론트가 아래)

가령 "낙시"라는 문제에서는  $N \leq 2000$  이기 때문에 배열의 크기를 2000보다 크게 잡아 줘야 하지만, 200으로 잡아 올바르지 않은 곳을 참조하는 코드이다. 따라서 Segmentation Fault가 나며, 실제로 서버에서도 SIGSEGV 시그널이 발생함을 알 수 있다. 위와 같은 코드는 런타임 에러를 발생시킨다.

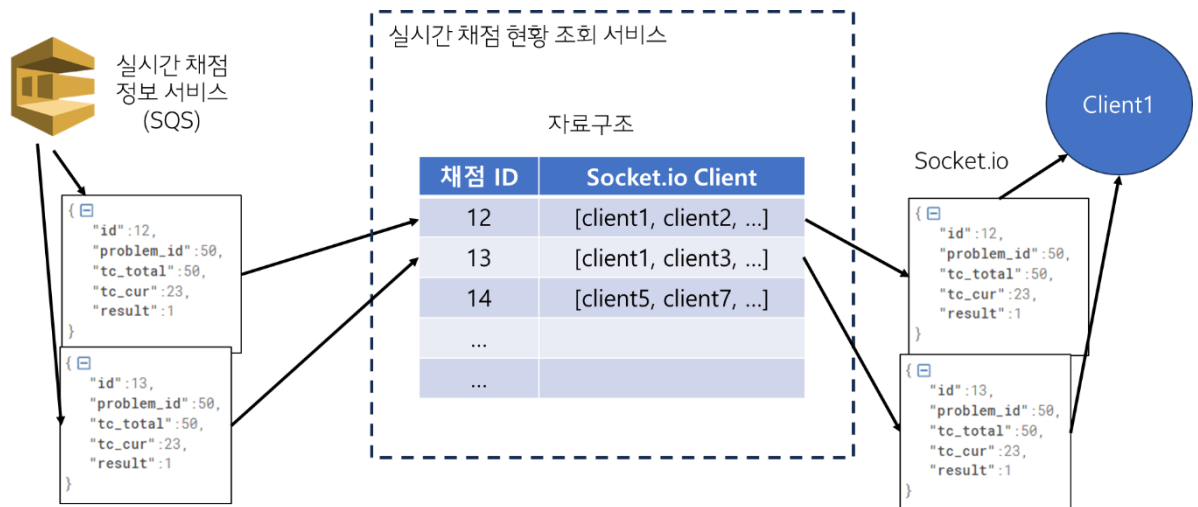
#### 4.6.7. Message Queues

서비스 간 통신을 위해 AWS SQS, SNS를 선택, 실제 사용에 대한 그림은 다음과 같다.



모든 서비스는 SQS로부터 메시지를 수신하고, SNS에 Topic을 발생시킨다. 각 SNS, SQS 간에는 Fanout Pattern에 따라, SQS가 SNS를 Subscribe하는 형태로 구성된다. 이로 인해 특정 이벤트가 이를 필요로 하는 처리자에게 복제되어 전달된다.

#### 4.6.8. Real Time Service

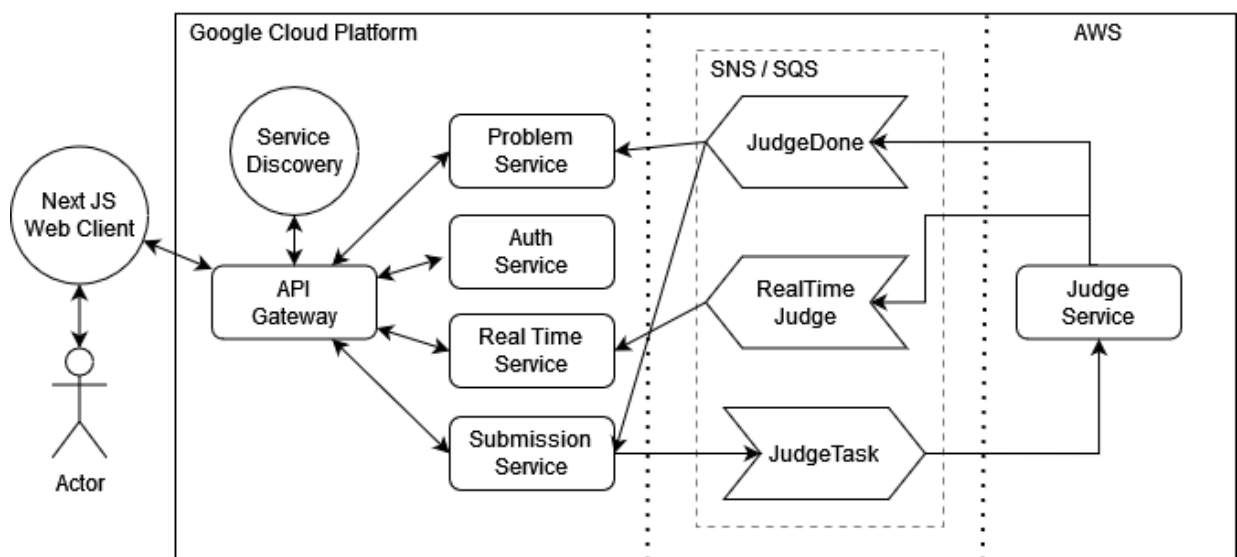


Real Time Service는 실시간 채점 결과를 Client에게 전달하기 위하여 새로 생성한 서비스이다. 이 서비스는 AWS의 SQS에 있는 RTQueue에서부터 채점 id, 테스트 케이스에 대한 정보를 전달받고, 내부적으로 저장 되어있는 자료 구조를 통해서, 특정 채점 id를 구독하고 있는 client들에게 Socket.io 라이브러리를 통해서, 테스트 케이스가 채점이 될 때 마다, 클라이언트로 채점 정보를 전달해 준다.

채점 정보에는 채점 ID, 전체 테스트 케이스 개수, 성공한 테스트 케이스 개수, 채점 상태코드가 전달된다. 만약 TC 채점이 성공적으로 완료되지 않은 경우, 더 이상 큐에 TC 채점 정보가 발생하지 않는다. 모든 TC 채점이 성공한 경우, 가장 오래 걸린 시간과 가장 많은 메모리 이용량을 반환하여 클라이언트에서 표시해 준다.

## 5. 최종산출물 형태

### 5.1 Deployment Diagram



모든 서비스들은 GCP Compute Engine 혹은 AWS EC2에 Deploy 된다.

서비스 간 통신에는 기본적으로 PaaS 형태로 제공되는 AWS의 SNS, SQS를 사용하게 된다.

그 외에 사용자의 요청을 처리하는 과정에서의 Front – Gateway – Services 로 HTTP 로 통신, Service – Service Discovery 간 Register, Healthcheck 목적의 HTTP 통신이 존재한다.

## 5.2. Distributed Layered Architecture

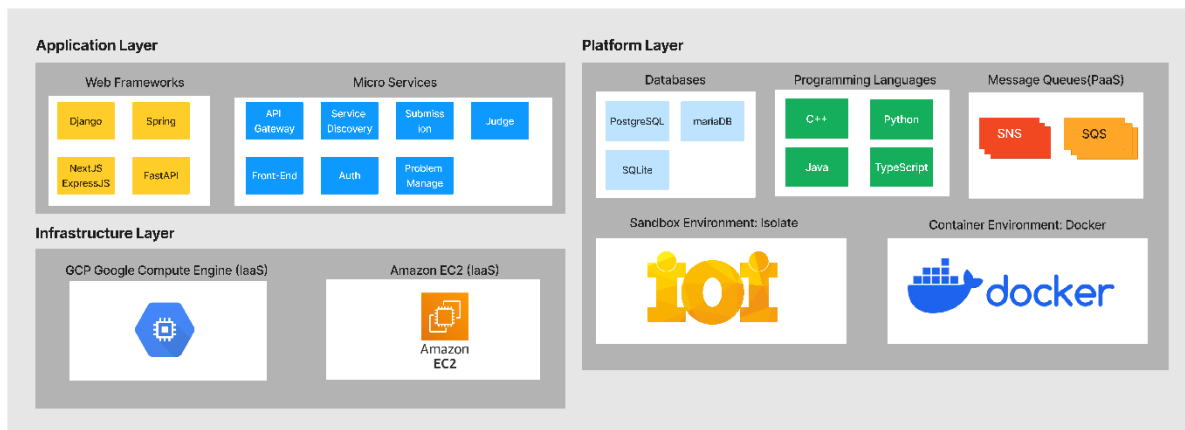


그림 13- Application/Platform/Infrastructure Layers

최종 산출물에 대한 Application, Platform, Infrastructure Layer는 위의 그림과 표현된다.

모든 Infrastructure는 IaaS 형태로 제공되는 GCP Google Compute Engine 혹은 AWS EC2 이다.

이러한 Infrastructure 위에 각 Application들이 의존할 Platform들이 존재한다. 이러한 Platform 들의 종류로는 Databases, Programming Languages, Sandbox Environment, Container Environment 가 존재한다. Infra 바깥에 PaaS 형태로 존재하는 것들도 있는데, AWS SQS, SNS 등이 이에 해당된다.

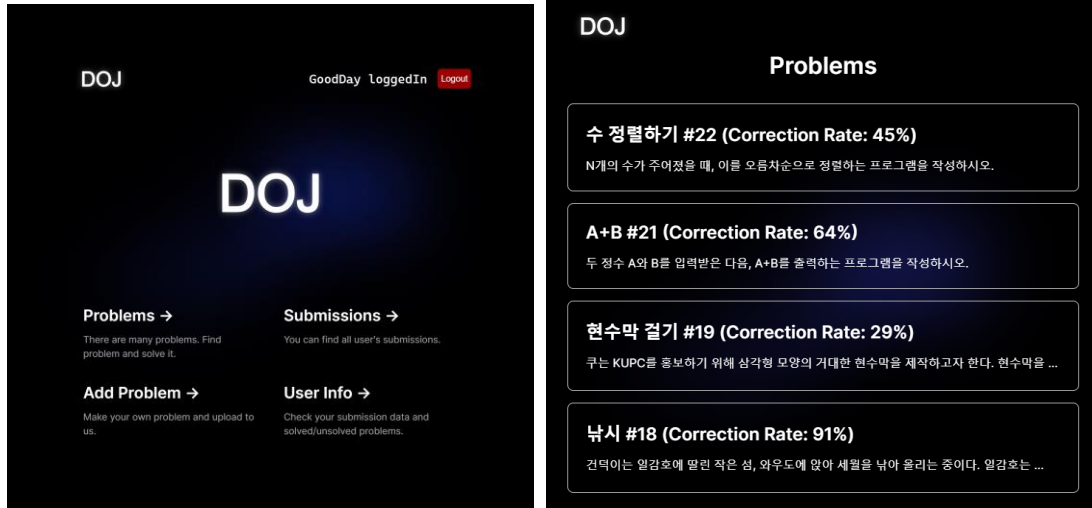
Application의 경우 직접 구현한 서비스들 및 그들이 의존하는 웹 프레임워크 등이 존재한다. Application 측 기술 스택의 경우 다음과 같은 표로 간략하게 정리하는 것이 가능하다.

Micro Services	Framework (Language)	DB
API Gateway	Spring Cloud Gateway (Java)	-
Service Discovery	Spring Cloud Netflix – Eureka Server	-
Submission Service	Django (Python)	PostgreSQL
Judge Service	Isolate + rapidjson (C++)	-
Auth Service	FastAPI (Python)	SQLite
Problem Manage Service	Spring Web MVC (Java)	MariaDB
Real Time Service	Express.js + Socket.io (Node.js)	-
Frontend Service	Next.js (Node.js)	-

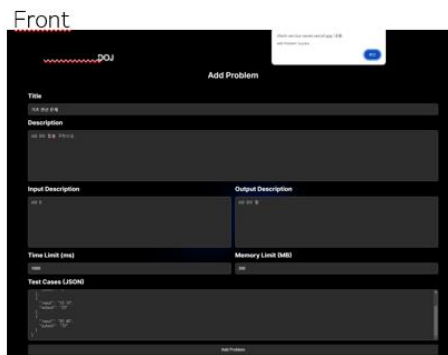
### 5.3 실제 결과물

배포 사이트 -> <https://client-service-seven.vercel.app>

#### 메인 화면 및 문제 목록 확인



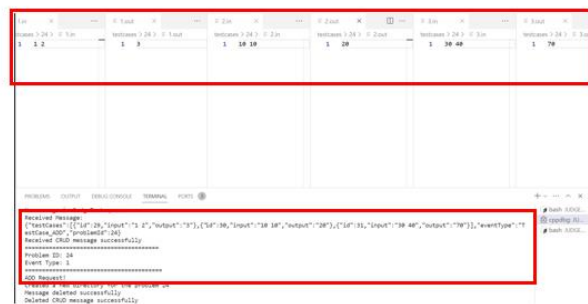
#### 문제 및 테스트케이스 생성



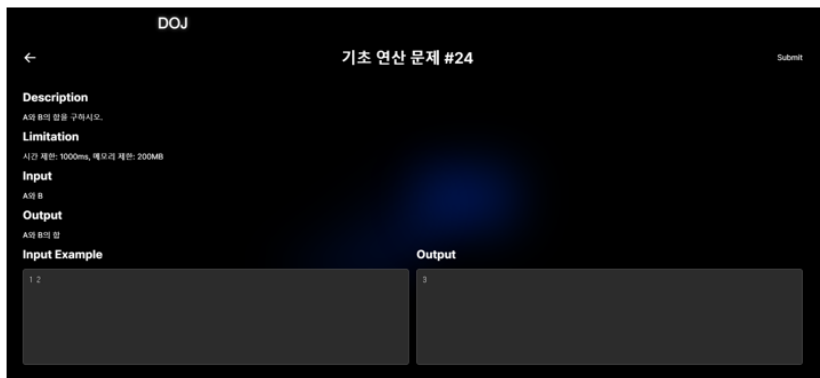
문제 관리 서비스 - 문제 발행

```
2023-12-17T00:04:22.254Z INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration  
Send Topic : {testCases:[{id=29, input='1 2', output='3'}, {id=30, input='10 10', output='20'}, {id=31, input='30 40', output='70'}], eventType=TestCase_ADD, problemId=24}
```

채점 서버 내 파일 동기화



#### 문제 조회



```
DOJ
Submission #890 (Problem #24, User #1)

Submit Code

1 #include<iostream>
2 using namespace std;
3
4 int main(void){
5     int a;
6     int b;
7     cin >> a >> b;
8     int answer = a + b;
9     cout << answer;
10    return 0;
11 }
```

Submitted At: 2023. 12. 17. 오후 6:33:33

Program Language: C++20

Result: Judging...



```

Submission service | Fetched From Queue of my own
Submission service | Message Sent >> ['id': 1012, 'problem id': 16, 'user id': 1, 'judge status': 0, 'source': '\n#include <bits/stdc++.h>\n#define fastio cin.tie(0) ->sync_with_stdio(0)\nusing namespace std;\nint N, M, W, P, ocean[1000000];\n2005);\n\nint main() {\n    fastio;\n    cin > N >> M >> Q;\n    for(int i = 0; i < N; i++)\n        for(int j = 0; j < M; j++)\n            cin >> ocean[i][j];\n    if(i > 0) ocean[i][j] >= ocean[i-1][j];\n    if(i < 0) ocean[i][j] <= ocean[i+1][j];\n    if(i > 1) ocean[i][j] <= ocean[i-2][j-1];\n    if(i < -1) ocean[i][j] <= ocean[i+2][j+1];\n    int cout << ocean[W-1][P-1] << "\\n\\n\\n";\n    while(Q--)\n        cin >> W >> P;\n    return 0;\n}\n// Language: C++\nCode: 1, 'judge description': 'Not judged', 'error message': 'Created time: 2023-12-17T22:24:07.653651+09:00', 'start time': '2023-12-17T22:24:07.653734+09:00', 'end time': None, 'memory used': -1, 'time used': -1, 'memory limited': 1024, 'time limited': 2000]

```

- 브로커

```
Received Message From JudgeTask:
{
  "Type": "Notification",
  "MessageId": "ff5a1eab-547a-59e3-a816-43525e7b2a26",
  "TopicArn": "arn:aws:sns:ap-northeast-2:262981387273:DOJ-Submission-SubmitTask.fifo",
  "Subject": "submit",
  "Message": "{\n    \"id\": 1138, \"problem_id\": 18, \"user_id\": 1, \"judge_status\": 0, \"source\": \"\\n\\n #include <bits/stdc++.h>\\n#define fastio cin.tie(0) ->sync_with_stdio(0)\\n\\nusing namespace std;\\n\\nint N, M, Q, W, P, ocean[2005];\\n\\nint main() {\\n    fastio;\\n    cin >> N >> M >> Q;\\n\\n for (int i = 0; i < N; ++i) {\\n    for(int j = 0; j < M; ++j) {\\n        cin >> ocean[i][j];\\n        if (i > 0) ocean[i][j] += ocean[i-1][j];\\n        if (i && j) {\\n            ocean[i][j] += ocean[i-1][j-1];\\n            if (i > 1) ocean[i][j] -= ocean[i-2][j-1];\\n        }\\n        \\n        \\n    }\\n    while(Q--) {\\n        cin >> W >> P;\\n        cout << ocean[W-1][P-1] << \"\\n\\n\\n\\n\\n\\n\\n\\n return 0;\\n}\\n    \\n\", \"language_code\" : 1, \"judge_description\": \"Not Judged\", \"error_message\": \"\", \"created_time\": \"2023-12-17T23:37:53.455358+09:00\", \"start_time\": \"2023-12-17T23:37:53.455437+09:00\", \"end_time\": null, \"memory_used\": -1, \"time_used\": -1, \"memory_limited\": 1024, \"time_limited\": 2000}, \"timestamp\": \"2023-12-17T14:37:53.642Z\", \"unsubscribeURL\": \"https://sns.ap-northeast-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:ap-northeast-2:262981387273:DOJ-Submission-SubmitTask.fifo:08e7ec0b-1f67-4c31-9021-208b47c74eb9\" }"}
}
Received C/C++ submission message successfully
Message delivered successfully to queue: JudgeCPP.fifo
Delivered message to queueJudgeCPP.fifo successfully
```

제출 정보와 문제 CRUD 요청을 받아서 처리  
제출 정보일 경우, 제출된 언어에 따라 다른 큐 (SQS)로 전달



## • 워커

```
Received Message:
{"id": 1006, "problem_id": 18, "user_id": 1, "judge_status": 0, "source": "\n #include <b
its/stdc++.h>\n#define fastio cin.tie(0)->sync_with_stdio(0)\nusing namespace std;\n\nint
N, M, Q, W, P, ocean[2005][2005];\n\nint main() {\n    fastio;\n    cin >> N >> M >> Q;\n\n    for(int i = 0; i < N; i++) {\n        for(int j = 0; j < M; j++) {\n            cin >> oc
ean[i][j];\n            if(i > 0) ocean[i][j] += ocean[i-1][j];\n            if(i && j) {\n
an[i-2][j-1];\n            ocean[i][j] += ocean[i-1][j-1];\n            while(Q-->) {\n                cin >> P;\n
cout << ocean[W-1][P-1] << "\\n";\n            }\n            return 0;\n        }\n    }\n    "language_code"
: 1, "judge_description": "Not Judged", "error_message": "", "created_time": "2023-12-17T23:
30:07.591579+09:00", "start_time": "2023-12-17T23:30:07.591678+09:00", "end_time": null, "me
mory_used": -1, "time_used": -1, "memory_limited": 1024, "time_limited": 2000}
Received the message successfully
Message deleted successfully.
```

제출 정보

```
Submission #1000 (Problem #18, User #1)
Submitted At: 2023. 12. 17. 오후 10:17:29
Program Language: C++20
Result: Accepted (Time: 230 ms, Mem: 61084 KB)

Submission #1001 (Problem #18, User #1)
Submitted At: 2023. 12. 17. 오후 10:18:03
Program Language: C++20
Result: Accepted (Time: 222 ms, Mem: 61096 KB)

Submission #1002 (Problem #18, User #1)
Submitted At: 2023. 12. 17. 오후 10:18:35
Program Language: C++20
Result: Judging... (38%)
```

여러 워커들이  
동시에 채점 중

```
=====
18 Statistics:
AC: 24
NOT AC: 0
Judge result: AC
Max Time: 276ms
Max Memory: 61108KB
Error Message:
=====
```

## 채점 결과 수신

```
submission_service Fetched From Queue of my own
submission_service Message Sent -> {'id': 1006, 'problem_id': 18, 'user_id
define fastio cin.tie(0)->sync with stdio(0)\nusing namespace std;\n\nint N, M, Q, W
M >> Q;\n\nfor(int i = 0; i < N; i++) {\n    for(int j = 0; j < M; j++) {\n        \n
ocean[i-1][j];\n        if(i && j) {\n            ocean[i][j] += ocean[i-1][j-1]
\n            while(Q-->) {\n                cin >> W >> P;\n                cout <<
code': 1, 'judge_description': 'Not Judged', 'error_message': '', 'created_time': '20
81749+09:00', 'end_time': None, 'memory_used': -1, 'time_used': -1, 'memory_limited':
submission_service Processing 1 Submissions..
submission_service Submission[1006] -> {'judge_status': 1, 'error_message
: None, 'judge_description': 'Accepted', 'time_used': 241, 'memory_used': 61148}
submission_service Updated 1 Submissions
submission_service Deleting 1 Items
```

채점 결과를 수신하여 제출의 상태 갱신

```
MessagePayload(Type=Notification, MessageId=de3a0332-f112-56d6-a713-5ee7b4c75191,
SequenceNumber=10000000000000000000, TopicArn=arn:aws:sns:ap-northeast-2:26298138
273:DOJ-Judge-JudgeDone.fifo, Subject=null, Message={'id':24,'user_id':7,'problem
id':18,'language_code':1,'memory_limited':100,'time_limited':1000,'judge_result':
,'error_message':null}, Timestamp=2023-11-20T11:31:33.319Z, UnsubscribeURL=https:
/sns.ap-northeast-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns
:ap-northeast-2:26298138273:DOJ-Judge-JudgeDone.fifo:70acb7b5-d171-410b-8516-d1ce
a471bf0f)
ResultSubmit(id=24, user_id=7, problem_id=18, language_code=1, memory_limited=100
time_limited=1000, judge_result=1, error_message=null)
2023-11-20T11:31:40.242Z INFO 1 (trap_executor 0) c.m.d.s.r.aws.ConfigCluste
revisions
```

채점의 결과를 수신하여 문제의 틀린 횟수, 맞은 횟수 갱신

## 제출 결과 확인

### DOJ

#### Submissions

**Submission #1002 (Problem #18, User #1)**

Submitted At: 2023. 12. 17. 오후 10:18:35  
Program Language: C++20  
Result: Judging... (38%)

**Submission #1001 (Problem #18, User #1)**

Submitted At: 2023. 12. 17. 오후 10:18:03  
Program Language: C++20  
Result: Accepted (Time: 222 ms, Mem: 61096 KB)

**Submission #1000 (Problem #18, User #1)**

Submitted At: 2023. 12. 17. 오후 10:17:29  
Program Language: C++20  
Result: Accepted (Time: 230 ms, Mem: 61084 KB)

### DOJ

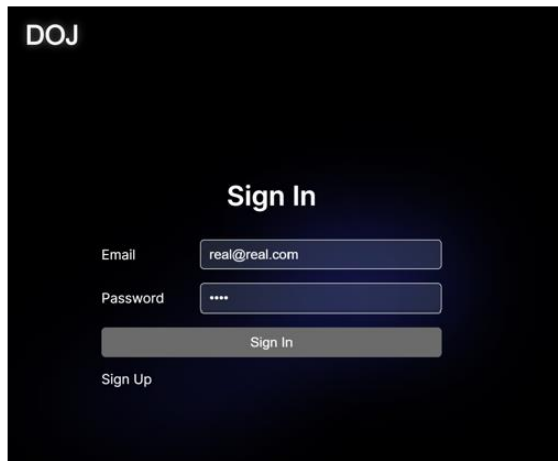
#### Submission #888 (Problem #21, User #1)

##### Submit Code

```
1 A, B = input().split() # 입력되는 문자를 input() 함수로 입력받고 sp
2
3 print(int(A)+int(B)) # int() 함수로 A와 B를 정수로 변환 하고 두 수
4
```

Submitted At: 2023. 12. 17. 오후 5:29:12  
Program Language: Python 3.10.9  
Result: Judging... (28%)

## 이용자 인증



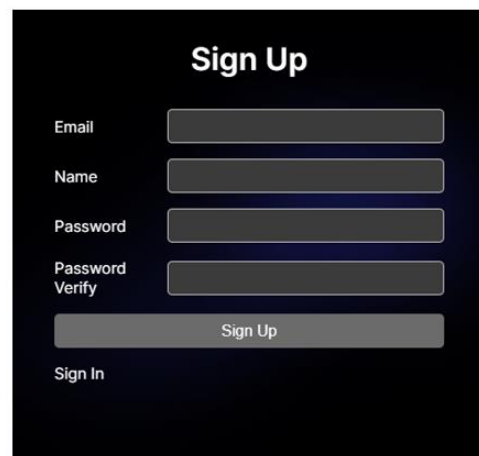
DOJ

### Sign In

Email:

Password:

[Sign Up](#)



### Sign Up

Email:

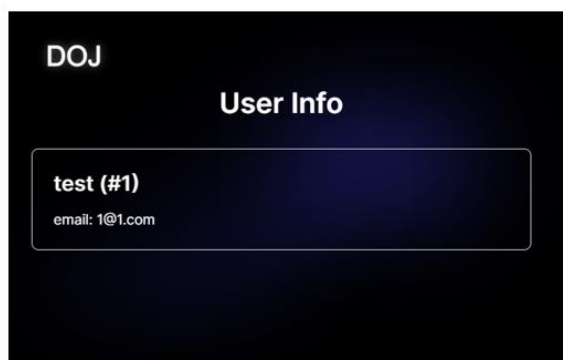
Name:

Password:

Password Verify:

[Sign In](#)

## 이용자 정보 조회

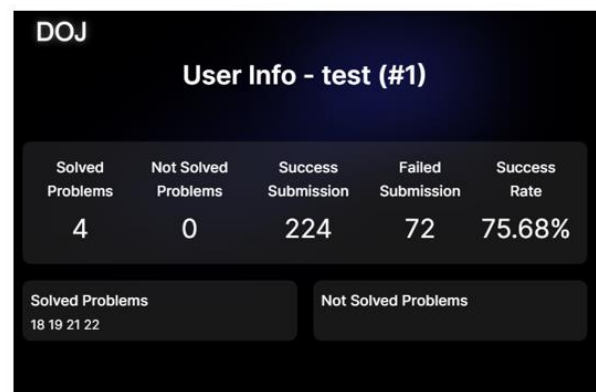


DOJ

### User Info

**test (#1)**

email: 1@1.com



DOJ

### User Info - test (#1)

Solved Problems	Not Solved Problems	Success Submission	Failed Submission	Success Rate
4	0	224	72	75.68%

Solved Problems  
18 19 21 22

Not Solved Problems

## 6. Risk Analysis & Risk Reduction Plan

### 6.1. 완료한 Risk 목록

현재까지 완료한 Risk 목록은 다음과 같이 정리할 수 있다.

- 마이크로 서비스 간 통신
- 격리된 채점 환경 구축
- 인증 관련 문제 해결
- 데이터 동기화
- 실시간 채점
- 채점 결과에 Memory Usage & Time Usage 기록

- 제출 발생에 대한 Throttling

각 내용에 대한 상세 설명은 다음과 같다.

- 마이크로 서비스 간 통신

서비스 간 통신에 메시지 큐를 사용하며, 이 과정에서 HTTP를 사용해서 통신하는 것에 비해 명확한 이점들이 존재한다. 이를 위해서 AWS에서 제공하는 SQS, SNS를 사용한다. FIFO가 보장되고, 관련 자료가 많으며 이를 사용하는 Fanout Pattern이 현 프로젝트의 방향과 부합하기 때문이다.

- 격리된 채점 환경 구축

isolate를 사용하여, 문제에서 제시한 시간/메모리 제한에 부합하는 격리된 환경에서 사용자의 코드를 실행할 수 있다.

- 인증 관련 문제해결

Microservice Architecture는 필연적으로 한 서버에 존재하는 DB의 세션 정보를 활용하는 것이 불가능하다. 따라서 JWT를 사용해서 서비스간 인증 관련 문제를 해결하였다. 모든 HTTP 요청에 JWT를 포함시켜, 인증 처리 및 사용자 정보 이용이 가능하다.

- 제출에 대한 Throttling

Judge Service로 시간 당 향하는 제출을 제한하는 Throttling을 구현, 의도한 것보다 많은 시간 당 제출이 발생할 경우 Submission Service는 이를 Redis에 기록하여, 순차적으로 내보내는 것으로 구현하였다.

- 데이터 동기화

서비스 간의 데이터 동기화를 제공하는 것이 필요하다. Judge Service 와 Problem-Manage Service 간에는 문제 정보, 테스트 케이스 정보에 대한 동기화를 필요로 하며, Judge Service와 Submission Service 간에는 채점 발생, 채점 완료에 대한 데이터의 동기화를 필요로 한다. 또한 Judge Service와 Real Time Service 간에 실시간 채점 데이터를 공유하는 과정에서도 데이터 동기화를 필요로 한다. 이 모든 서비스 간 데이터 동기화는 SQS, SNS를 사용하여 제공하고 있다.

- 실시간 채점

Real Time Service는 실시간 채점 결과를 Client에게 전달하기 위하여 새로 생성한 서비스이다. 이 서비스는 AWS의 SQS에 있는 RTQueue에서부터 채점 id, 테스트 케이스에 대한 정보를 전달받고, 내부적으로 저장 되어있는 자료 구조를 통해서, 특정 채점 id를 구독하고 있는 client들에게 Socket.io 라이브러리를 통해서, 테스트 케이스

가 채점이 될 때 마다, 클라이언트로 채점 정보를 전달해 준다.

채점 정보에는 채점 ID, 전체 테스트 케이스 개수, 성공한 테스트 케이스 개수, 채점 상태코드가 전달된다. 만약 TC 채점이 성공적으로 완료되지 않은 경우, 더 이상 큐에 TC 채점 정보가 발생하지 않는다. 모든 TC 채점이 성공한 경우, 가장 오래 걸린 시간과 가장 많은 메모리 이용량을 반환하여 클라이언트에서 표시해 준다.

- 채점 결과에 Memory Usage & Time Usage 기록

Judge Service에서는 채점 중 발생한 Time Usage & Memory Usage의 추적이 가능하다. 정확히는 채점에서 확인하는 Test Case 마다 Time Usage & Memory Usage를 발생시키는데, Judge Service는 이 중 가장 크게 발생한 Time Usage & Memory Usage를 내보낸다. 즉 가장 오래 걸리고 메모리를 많이 쓴 것들만 메시지 큐를 통해 내보내어 Submission Service 등에서 기록하도록 한다.

## 6.2. 해결할 Risk 목록

- 채점 환경의 브로커, 워커 머신간 분리

현재 채점 환경은 브로커와 워커가 같은 머신에 존재함을 가정한 채 개발되어, 서로 결합되어 있는 상태이다. 이로 인해 테스트 케이스 정보를 파일 시스템에 직접 접근하여 사용하게 되어 있다. 즉 채점을 하는 워커들을 여러 머신에서 분산되어 실행하고자 할 때에는 워커 / 브로커 간의 테스트 케이스 동기화가 필요하다. 이들의 결합성을 줄이고 다른 머신들에 대해서 Scale Up을 제공하려면 Test Case를 관리하는 Cache Server 를 구축하여야 한다.

## 7. Success Criteria & Test

완성된 서비스에 대해서는 다음과 같은 성공 기준을 마련하였다.

- 일관성

본 서비스는 동일한 제출(동일한 코드, 동일한 언어, 동일한 문제)에 대해서 일관성 있는 결과를 제공해야 한다. 이때 일관성 있는 결과란, 채점에서 내보내는 Memory Usage & Time Usage가 크게 변하지 않는 것을 말한다. 이를 만족해야 일관성을 제공한다고 정의했다.

- 부하

본 서비스는 많은 양의 요청을 감당할 수 있어야 한다. 이를 위해 채점 서비스의 양을 늘려 병렬적으로 처리할 수 있도록 하였고, 제출 서비스에서도 Throttling을 구현한 것이기 때문이다. 따라서 여러 요청이 동시다발적으로 발생했을 때 서비스 제공은 문제 없이 지속됨을 보여야 한다.

그에 대한 테스트는 다음과 같이 작성하였다.

- 테스트 방법

Locust를 통해 동일한 제출을 동시다발적으로 요청(10 Request/Sec) 시킨다. 이후 해당 제출에 대한 결과가 일정 시간(30s) 내에 의도한 대로 채점 완료되었는지를 확인한다.

- 실제 테스트 코드

```
@task
def post_and_compare(self):
    submit_id = None
    with self.client.post(...)
        res_json = res.json()
        if not res.ok: ...

        res_json = res.json()
        data = res_json.get("data")
        if data is None: ...

        submit_id = data.get("id")
        if submit_id is None: ...
        print(f"Waiting for -> {submit_id}")
        time.sleep(30)
        print(f"Waiting Done -> {submit_id}")

        print(f"Retrieve Result and Compare")
        g_res: models.Response = self.client.get(...)
        if not g_res.ok: ...

        g_json = g_res.json()
        g_data = g_json.get("data")
        if g_data is None:
            res.failure("detail data is None")

        g_submit_id = g_data.get("id")
        if g_submit_id is None: ...

        if g_submit_id == submit_id:
            judge_status = g_data.get("judge_status", 0)
            print(f"judged status -> {judge_status}[{submit_id}]")
            print(g_data)
            if judge_status == 1:
                print(f"Success for {submit_id}")
            else:
                res.failure("non success")
```

테스트 코드에서는 POST 요청을 통해 제출을 발생시키고, 그것의 ID를 기억, 30초간 대

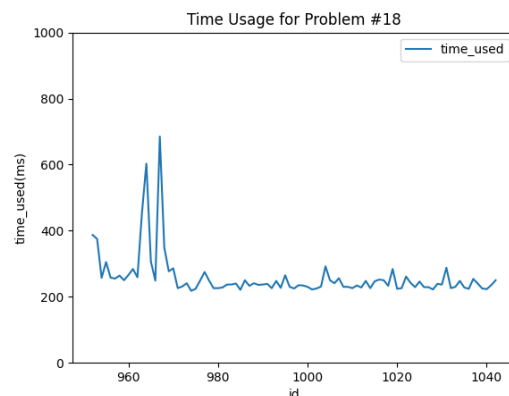
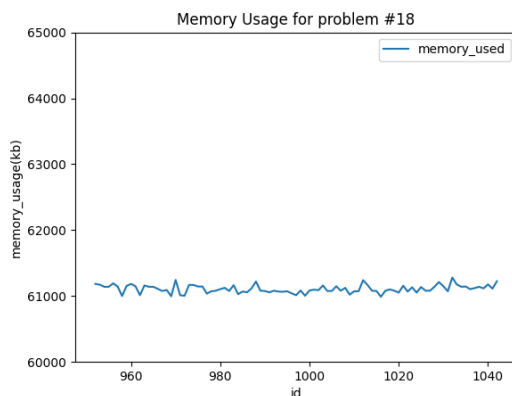
기한 후 해당 제출의 상태를 확인하여 채점이 성공하였는지를 확인하는 Task를 실행한다.

이에 대한 결과는 Locust를 통해 확인할 수 있다.

Statistics Charts Failures Exceptions Current ratio Download Data												
Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/submission_service /submit/18	97	9	290	370	700	301	224	702	343	0	0
Aggregated		97	9	290	370	700	301	224	702	343	0	0

97개 중 9개 정도의 제출이 30초 내에 통과하지 못한 것을 확인할 수 있으며, 그 동안 전체 서비스의 제공에는 큰 문제가 존재하지 않았다.

이때 발생한 제출들에 대한 Memory Usage, Time Usage는 다음의 그림으로 표현 가능하다.



메모리 사용량의 경우 61000 kb ~ 62000 kb 사이에서 일관된 결과를 보이고 있으며, Time Usage의 경우 200ms ~ 800ms 사이의 결과를 보이고 있다.

서비스 제공에 지장이 없으며, 제출 결과는 의도한 시간 제한, 메모리 제한 내에서 Accepted 가능한 값을 보이고 있기에 Test를 성공했다고 판단했다.

## 8. 최종 결과 및 분석

최종적으로 구현한 서비스는 다음과 같다:

- 문제 관리 서비스
- 제출 서비스
- 채점 서비스

- 실시간 채점 현황 서비스
- 인증 서비스

서비스들은 EC2와 GCP VM 등 다양한 환경에서, Cpp와 Spring, Django, Next.js 등 다양한 프로그래밍 언어 및 프레임워크들로 개발되었다. 역할분담을 서비스 단위로 나누었기 때문에, 병렬적 개발을 통해 개발에 따른 리소스 소모를 최소화할 수 있었다. 서비스에 이슈가 발생되면 독립된 서비스 컨테이너의 로그를 확인하고 수정하여 효율적인 유지보수를 할 수 있었다. 채점 결과와 테스트 케이스 등 서비스 간 동기화가 필요한 데이터는 AWS SQS/SNS를 통해 메시지를 발행 및 폴링하여 동기화된다. 모든 서비스들은 도커 컨테이너를 통해 배포되기 때문에 도커를 지원하는 모든 OS에서 서비스를 배포할 수 있다. 또한 채점 작업량이 증가함에 따라 채점 서비스를 스케일 아웃하여 부하를 분산한다. 따라서, MSA의 장점을 극대화하여 짧은 시간동안 온라인 저지 서비스를 개발할 수 있었다.

## 9. 상세 스케줄과 상세 역할 분담

제목	시작일	종료일	내용
프로젝트 제안서 작성	10/23	10/30	프로젝트 제안서 작성
API 기능 정의 및 명세서 작성	11/5	11/12	프로젝트 요구사항에 따른 서비스별 API 기능 정의 및 명세서 README.md 작성
Gateway 테스트 완료	11/12	11/14	배포 서비스를 Spring Cloud Discovery와 Spring Cloud Gateway를 통해 API 등록
문제 및 테스트케이스 기능 초기 구현	11/12	11/19	초기 테스트를 위한 문제 및 테스트케이스 API 서비스 개발
제출 서비스 초기 구현	11/12	11/19	채점 테스트를 위한 답안 제출 서비스 개발
채점 서비스 초기 구현	11/12	11/19	정답/오답 구분 가능한 Cpp 채점 서비스 개발
프론트 서비스 초기 구현	11/12	11/19	API 요청을 통해 문제 CRUD와 답안 작성, 제출 및 채점 기능 구현
인증 관련 문제 해결	12/1	12/4	HTTP 요청에 JWT 포함
채점 기능 보완	12/1	12/5	제출 정보에 사용한 시간 및 메모리 정보 포함
실시간 채점 현황 구현	12/4	12/14	SQS를 통해 매 테스트케이스 채점 시 그 결과를 큐로 전달
데이터 동기화	12/3	12/10	채점 서비스와 문제 관리 서비스 간의 데이터 동기화 구현

제출 Throttling 구현	12/4	12/15	많은 제출이 발생하면 Redis에 기록하여 순차적으로 내보냄
채점 서비스 분리 및 컨테이너화	12/15	12/16	워커와 브로커로 분리 및 각각에 대한 도커 이미지 빌드 완료
채점 서비스 클라우드에 배포	12/15	12/18	GCP에 채점 서비스 배포 완료

번호	이름	학번	역할	역할 상세
1	백종원	201811263	백엔드	- 제출 서비스
2	김창엽	201811244	채점 엔진	- 채점 서비스
3	정우철	201811293	백엔드 프론트엔드	- API Gateway & Eureka - 사용자 서비스 - 실시간 채점 서비스 - Frontend
4	강병우	201911146	백엔드	- 문제관리 서비스

## 참고 자료 목록

API Gateway & Auth Service

- [API Gateway & Opaque Token Example](#)

Message Queue

SQS, SNS

- [SQS 소개](#)
- [SNS 소개](#)
- [SNS Documentation](#)
- [SQS Documentation](#)

Fanout Pattern

- [Fanout Pattern \(AWS Documentation\)](#)
- [Fanout Pattern 관련 정리\(한글\)](#)

Judge Service

샌드박스 관련

- [Mareš, M. et al. \(2012\) A New Contest Sandbox. \*Olympiads in Informatics\*.](#)



- [IOI Technical Checklist - Determinism](#)
- [IOI. ISOLATE MANUAL](#)
- [Baekjoon Choi \(2015\) 샌드박스](#)

채점 서버 관련:

- BOJ 10 Years – [1부](#) (아키텍처), [2부](#) (채점 프로세스) – (2019). Startlink.io
- BOJ Help – [언어 정보](#), [채점 정보](#)
- 다양한 온라인 저지들: [BOJ](#), [HUSTOJ](#), [DMOJ](#),
- JSON 파싱 라이브러리: [rapidjson](#)

Submission Service

- [Starting Django](#)
- [AWS SDK for Python; boto3](#)
- [Celery + Python](#)
- [Throttling via Redis](#)

Problem-Manage Service

- [Spring JPA](#)
- [AWS Sqs with AWS SDK For Java](#)
- [AWS Sns with AWS SDK For Java](#)