

AWS 기반 글로벌 저지연 스트리밍 서비스

인프라 구축과 애플리케이션 배포 검증

Team awesome

조현수

4인 팀 프로젝트 · 본인 담당 파트 발췌

본 포트폴리오는 4인 팀 프로젝트 중 본인 담당 영역을 중심으로 정리하였습니다.

프로젝트 개요 · 담당 영역

서비스

음원 업로드 · 재생 스트리밍
회원 인증 · 좋아요 · 팔로우 · 알림 · 다국어

개발 영역

Node.js + Express 백엔드 · React 프론트엔드
MySQL 데이터 저장 · nginx 웹 서버

인프라 영역

서울 · 상파울루 2개 리전 · VPC 3계층
ALB · Auto Scaling · RDS Multi-AZ · S3

본인 담당

- ✓ 서버 내부 구축(런타임 · 웹 서버 · 프로세스 관리)
- ✓ 앱 검증을 위한 AWS 환경 구성 및 배포 테스트
- ✓ 정상 구동 검증 및 절차 정리
- ✓ 애플리케이션 확장 및 기능 구현

팀 담당

- ✓ 전체 인프라 아키텍처 설계
- ✓ 인프라 구축 방법 정립 및 표준화
- ✓ CloudFormation 기반 배포 자동화
- ✓ Global Accelerator · CloudFront WAF · 모니터링 설계
- ✓ 애플리케이션 초기 구조 설계 및 구현

EC2에서 직접 실행하며 정리한 서버 배포 절차가 팀 Launch Template UserData 자동화의 기반이 됨
초기 구조 이후의 애플리케이션 기능 개발 대부분을 AI 도구로 구현 · 검증

진행 타임라인

개발 중인 앱을 정해진 인프라에 올려 정상 동작을 반복 검증하였습니다.

앱 개발 시작
로컬 환경 구동 확인

AWS 배포 검증 시작
앱 배포 · 오류 확인 반복

서버 내부 구축 절차 정리

앱 개발 완료
최종 배포 검증

본인

팀

가상 네트워크 인프라 설계

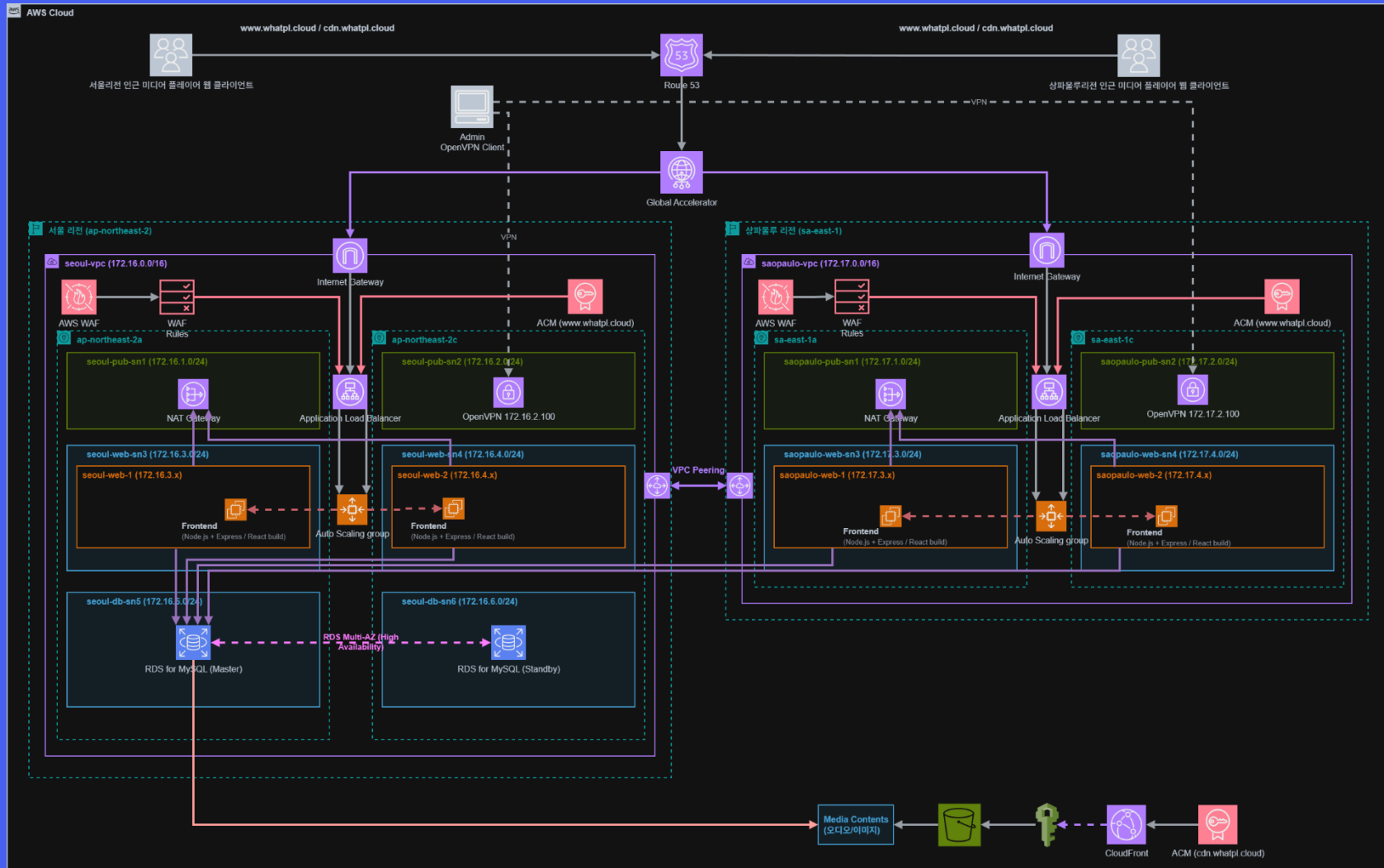
데이터 관리 · 백업 구성
부하분산 · 무중단 확장

자동화 스크립트 작성

글로벌 라우팅 · 가속
웹 방화벽(WAF) 적용

배포 자동화
권한 관리
리소스 모니터링

프로젝트 구성도



1. 앱 검증 환경 구성

설계된 구성도에 따라 검증 환경을 콘솔에서 직접 구축

네트워크

- VPC
- Subnet · 3계층 분리
- IGW · NAT Gateway
- Security Group
- OpenVPN
- Global Accelerator

컴퓨팅

- EC2
- Launch Template
- Auto Scaling
- ALB · Target Group

DB · 스토리지

- RDS for MySQL
- S3 Bucket

설계안과 동일한 구성을 반복 구축하며 앱 배포와 구동을 검증하였습니다.

2. 서버 내부 구축 및 배포 절차

EC2에 SSH로 접속해 절차를 직접 실행하며 정리 → 팀에서 Launch Template UserData로 자동화

① 런타임 설치 (Amazon Linux 2023)

dnf update -y

설치된 패키지를 최신 상태로 갱신하여 의존성 충돌 예방

dnf install -y git nginx unzip wget tar gzip

웹 서버와 파일 관리에 필요한 기본 유틸리티 일괄 설치

**curl -fsSL https://rpm.nodesource.com/setup_20.x | bash -
&& dnf install -y nodejs**

기본 nodejs 패키지가 18 버전이어서 NodeSource 저장소를 등록해 20.x 설치

npm install -g pm2

백엔드 상시 구동을 위한 PM2 프로세스 매니저 전역 설치

② 소스 코드 배치

wget -O whatpl.zip

**https://github.com/bluezeppelin/WHATPL/archive/
refs/heads/main.zip**

GitHub 원격 저장소에서 최신 코드를 압축 파일로 다운로드

unzip -q whatpl.zip

다운로드한 소스 압축 해제

mv WHATPL-main /opt/whatpl

서비스 운영 디렉터리(/opt/whatpl)로 이동하여 배치

전체 절차

① 런타임 설치 → ② 소스 배치 → ③ 환경 변수 구성 → ④ 빌드 · 프로세스 기동
→ ⑤ 웹 서버 설정 · 동작 확인

2. 서버 내부 구축 및 배포 절차

③ 환경 변수(.env) 구성 · ④ 빌드 및 프로세스 기동

③ 환경 변수(.env) 구성

데이터베이스 접속 정보

RDS 엔드포인트와 계정 · 비밀번호, 포트(3306) 입력

스토리지 · 구동 설정

S3 버킷 이름과 리전, 백엔드 포트(5000), 실행 모드(production) 지정

인증 키 설정

토큰 서명 키 지정 (자동화 스크립트는 기동 시 무작위 생성)

`chmod 600 /opt/whatpl/backend/.env`

DB 접속 정보가 포함되므로 소유자만 읽고 쓸 수 있도록 권한 제한

④ 빌드 및 프로세스 기동

PM2 도입 배경

node server.js로 직접 실행하면 접속 종료와 함께 프로세스도 종료됨

`npm ci --omit=dev`

운영에 필요한 백엔드 의존성 패키지만 설치

`pm2 start server.js`

백엔드를 상시 구동 상태로 기동

`pm2 startup && pm2 save`

서버 재부팅 시 백엔드가 자동으로 실행되도록 등록

`npm ci && npm run build`

프론트엔드 패키지 설치 및 배포용 정적 파일 빌드

2. 서버 내부 구축 및 배포 절차

⑤ 웹 서버(nginx) 설정 및 동작 확인

⑤ 웹 서버(nginx) 설정

라우팅 규칙 작성

/api/ → 백엔드(5000), /health → 백엔드(로드밸런서 상태 검사), 나머지 → 프론트엔드 빌드 파일

SPA 라우팅 · 업로드 용량 대응

없는 경로는 index.html 반환(try_files), 업로드 허용 크기 상향(client_max_body_size) — 트러블슈팅 사례 1 · 2

systemctl enable nginx && systemctl restart nginx

부팅 시 자동 실행 등록 및 재시작으로 설정 반영

동작 확인 (오른쪽 화면)

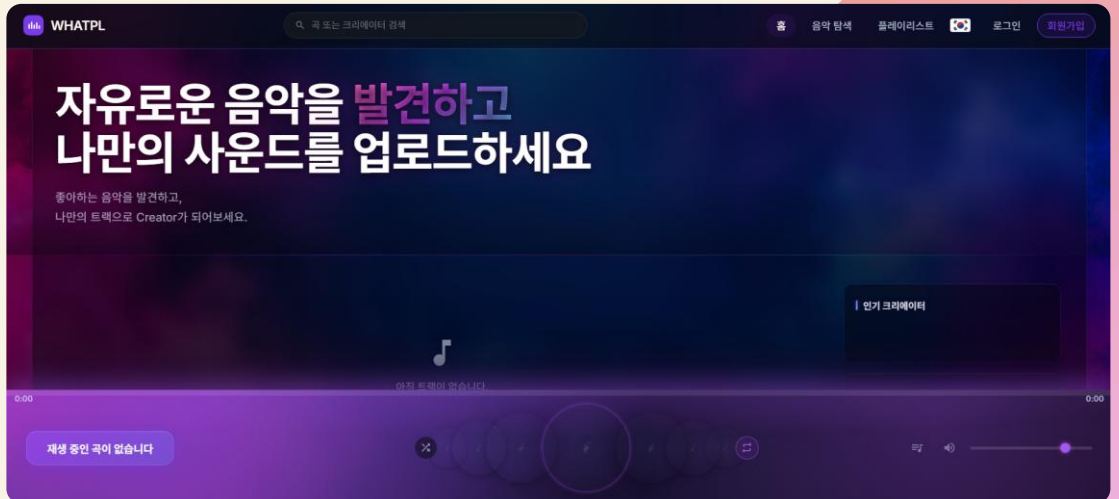
PM2 online · 재시작 0회, nginx -t 통과, nginx 경유 /health 200, 없는 경로도 200(index.html 반환)

id	name	namespace	version	mode	pid	uptime	cpu	status	cpu	mem	user	watching
0	whatpl-backend	default	1.0.0	fork	30479	2m	0	online	0%	60.9mb	root	disabled

```
host metrics | cpu: 1% | ram usage: 13.7% | ens5: 0mb/s | 0.001mb/s | disk: 0mb/s | 0.007mb/s
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful

HTTP/1.1 200 OK
Server: nginx/1.30.5
Date: Thu, 15 Oct 2026 03:17:29 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 15
Connection: keep-alive
X-Powered-By: Express
Vary: Origin
ETag: W/"f-VaSQ4oDUizbLZNAEkKn+sX+q3Sg"
{"status": "ok"}

no-such-page -> HTTP 200
```



배포 절차를 동일하게 재현하여 확인 (위: 서버 상태 · 아래: 서비스 첫 화면)

3. AI 도구를 활용한 애플리케이션 구현

개발 경험이 없는 상태에서 기능 정의와 검증을 직접 수행

본인

기획과 검증

- 필요한 기능을 정의하고 구현 순서를 결정
- 로컬 환경에서 동작을 확인하며 개발 진행
- 실행 결과를 보고 오류와 미흡한 부분을 판단
- 수정 요청을 반복하며 기능을 하나씩 완성
- 데이터 저장 방식을 JSON 파일에서 MySQL로 전환

AI 도구

코드 작성

- 백엔드 API와 프론트엔드 화면 코드 작성
- 데이터베이스 스키마 및 쿼리 작성
- 기능 추가 및 오류 수정

데이터 구조 전환 — Auto Scaling으로 서버가 여러 대가 되면서, 모든 서버가 같은 데이터를 참조하도록 JSON 파일에서 MySQL(RDS)로 전환

구분	JSON 파일	MySQL (RDS)
서버 2대 이상 운영 시	서버 간 데이터 불일치	모든 서버가 동일 데이터 참조
서버 교체 시	데이터 유실	유지됨
데이터 간 관계 (좋아요 · 팔로우 · 알림)	파일 내부에서 직접 관리	외래키로 관리

코드는 직접 작성하지 않았으나, 무엇을 만들지 정의하고 결과를 판단하는 과정을 담당하였습니다.

트러블슈팅

구축과 배포 과정에서 발생한 문제를 증상부터 조치까지 정리하였습니다.

구분	사례 1 · 새로고침 404	사례 2 · 업로드 용량 초과	사례 3 · 패키지 설치 미완료	사례 4 · S3 인증 정보 누락
증상	경로로 직접 접근하거나 새로고침하면 404 발생	음원 업로드가 실패함	t3.micro에서 UserData의 패키지 설치가 완료되지 않음	배포 후 S3 업로드가 동작하지 않음
확인 방법	nginx 오류 로그 확인 (/var/log/nginx/error.log)	크롬 개발자 도구에서 업로드 요청 응답 확인	SSH 접속 후 UserData 실행 로그 확인	크롬 개발자 도구에서 업로드 요청 응답 확인
원인	화면 경로를 브라우저(React)가 처리하는 구조라, 서버에는 해당 경로의 파일이 없음	nginx 요청 본문 크기 제한에 걸림 client_max_body_size 기본값 1MB	메모리 부족 추정	로컬 .env의 액세스 키 항목이 서버에는 없었음
조치	없는 경로는 index.html을 반환하도록 nginx 설정	50MB 음원 기준으로 허용 크기 상향	문제를 발견해 팀 회의에 공유하고 논의를 거쳐 t3.medium으로 상향 → 변경 후 설치 완료 확인	액세스 키 대신 EC2 IAM 역할 (인스턴스 프로파일)을 사용하도록 변경

회고 및 개선 계획

반복 작업의 자동화

앱을 올릴 때마다 같은 절차를
수동으로 반복하였음
초기에 스크립트로 묶었다면
검증 횟수를 더 늘릴 수 있었음

절차 공유 방식

정리한 내용을 문서 형태로 공유하였음
이후 실무실습에서 Ansible
플레이북으로 서버 배포 자동화 적용

환경 변수 구성 재검토

배포 스크립트(UserData)가 서버마다
서명 키를 새로 생성함을 확인
다중 서버 · 교체 시 인증 실패 가능,
Secrets Manager로 공유 필요