캡스톤디자인I 중간보고서

<table>
<tr><td rowspan="2">제 목</td><td colspan="2">국문</td><td colspan="3">AWS 활용 가상 머신 네트워크 시스템 구축 및 취약점 분석과 대응 솔루션 개발</td></tr>
<tr><td colspan="2">영문</td><td colspan="3">Building a Virtual Machine Network System Using AWS and Developing Vulnerability Analysis and Response Solutions</td></tr>
<tr><td rowspan="2">진 행
상 황</td><td>중요마일스톤</td><td colspan="4">1. 기능 요구사항
1) 웹 터미널 기능 정의 및 접속 흐름 명세 - 시나리오 명세서
2) Pod 생성 및 삭제 기능 설계 - 코드
3) 실시간 WebSocket 통신 구조 설계 - 코드
2. 인터페이스 요구사항
1) 프론트-백엔드-가상머신 간 통신 프로토콜 정의 - presignedURL
3. 보안 요구사항
1) PresigendURL, JWT, REDIS
4. 품질 요구사항
1) Kubernetes 기반 구조 설계 - 시스템 구성도, 아키텍처 그림
5. UI 요구사항
1) 피그마 사진 캡처</td></tr>
<tr><td>진행상황</td><td colspan="4">1. 웹 터미널 기능 : 코드 구현 완료 및 시행 구현
2. Pod 생성 / 삭제 : 코드 구현, 테스트 성공, 시나리오 작성
3. WebSocket 터미널 통신 : 프론트-가상머신 서버 간 실시간 통신 성공(로컬), 줄바꿈 등의 관련 처리 완료
4. JWT 기반 인증 및 redis 세션 만료 : 로그인 구현 완료, 유효시간 관리 및 토큰 만료 처리 구현
5. Presigend URL 발급/ 검증 : 백엔드에서 발급, 가상 머신 서버에서 검증 로직 작성 완료
6. UI 프로토타입 : 로그인, 회원가입, 서버 관리 UI 완료 및 적용</td></tr>
<tr><td>산출물</td><td colspan="5">요구사항 정의서(별첨 1), 중간보고서(별첨 2)</td></tr>
<tr><td rowspan="4">팀
구성원</td><td>학년</td><td>학 번</td><td>이 름</td><td colspan="2">연락처(전화번호/이메일)</td></tr>
<tr><td>4</td><td>20222517</td><td>한하영</td><td colspan="2">010-3068-3438 /
20222517@edu.hanbat.ac.kr</td></tr>
<tr><td>4</td><td>20222021</td><td>김주령</td><td colspan="2">010-7375-6602 /
20222021@edu.hanbat.ac.kr</td></tr>
<tr><td>4</td><td>20201738</td><td>서민재</td><td colspan="2">010-2750-5716 /
20201738@edu.hanbat.ac.kr</td></tr>
</table>

컴퓨터공학과의 프로젝트 관리규정에 따라 다음과 같이 요구사항 정의서와 중간보고서를 제출합니다

2025 년 5 월 2 일

책임자 : 서 민 재 (인)

지도교수 : 김 태 훈 (인)

[별첨1]

프로젝트명 : AWS 활용 가상 머신 네트워크 시스템 구축 및 취약점 분석과 대응 솔루션 개발

소프트웨어 요구사항 정의서

Version 1.0

개발 팀원 명(팀리더):한하영
서민재
김주령

대표 연락처:000-2750-5716
e-mail: 20201738@edu.hanbat.ac.kr

목차

1. 시스템 개요

가. 사업의 목표

1) 웹 기반 1회용 가상 머신 서비스 개발

가) 가상 머신 플랫폼의 필요성

가상화 기술은 하나의 물리 서버에서 여러 운영체제를 동시에 실행할 수 있도록 하여 자원 활용 효율성과 관리 편의성을 크게 향상시켰다. 기존의 물리적 설치 방식보다 유연하고 확장성 있는 환경을 제공하며, 서버 인프라 관리에 필수적인 기술로 자리잡았다.

나) 컨테이너 기반 가상화 기술의 활용

컨테이너 기술은 VM보다 가볍고 빠른 실행이 가능하며, 동일한 운영환경을 손쉽게 복제할 수있는 장점을 가진다. 특히 Kubernetes를 통해 자동 배포 및 확장이 가능하여 대규모 서비스 운영에 적합하다. 이러한 기술은 자원 최적화와 운영 효율성을 동시에 확보할 수 있다.

다) 기존 웹 가상 머신 서비스의 한계 극복

현대의 디지털 환경은 다양한 애플리케이션을 빠르게 배포하고 확장할 수 있는 유연한 인프라를 요구한다. 이에 따라 등장한 컨테이너 기술은 애플리케이션과 그 의존성을 패키징하여 일관된 환경에서 실행 가능하게 함으로써, 서버 관리의 효율성을 크게 향상시켰다. 또한 Kubernetes와 같은 컨테이너 오케스트레이션 도구를 활용하면 대규모 환경에서도 자동 배포 및 확장이 가능하여, 안정적이고 확장성 높은 서비스 운영이 가능하다.

라) 기존 웹 가상 머신 서비스의 한계 극복

기존의 가상 머신 및 컨테이너 기반 서비스는 다양한 가상 환경을 제공하지만, 설정의 복잡성, 성능 제한, 높은 비용 등의 문제로 일반 사용자의 접근성이 낮다. 로컬 VM은 하드웨어 의존도가 높고, 클라우드 VM은 초기 설정과 관리가 까다롭다. 또한 컨테이너 기반 환경은 운영체제 전체 실행에는 한계가 있으며, Kubernetes 등 오케스트레이션 도구는 구조가 복잡해 학습 곡선이 크다는 문제가 있다.

마) 서비스 제작 목표

기존 웹 가상 머신 서비스의 복잡성을 해결하고, 보다 간편한 사용성을 제공하는 웹 기반 1회용 가상 머신 서비스를 구축하고자 한다. 사용자는 별도의 설정 없이 원하는 운영체제를 즉시 실행하고, 일정 시간 후 자동 종료되도록 하여 자원 낭비를 최소화한다.

사용자의 편의성과 보안성을 고려하여 리소스 사용량 실시간 모니터링과 접속 제한(IP 필터링), 사용 이력 저장 및 복원 기능을 제공한다. 이를 통해 간편함, 확장성, 보안성을 동시에 갖춘 서비스를 구현한다.

나. 추진 범위

1) 사용 기술

본 프로젝트는 웹 기반 1회용 가상 머신 서비스를 구축하기 위해 다양한 최신 기술 스택과 클라우드 인프라를 활용한다. 프론트엔드는 React.js를 기반으로 구성하여 직관적인 사용자 UI를 제공하며, WebSocket을 통해 실시간 터미널 연결 기능을 지원한다. 백엔드는 Spring Boot를 중심으로 사용자 요청을 처리한다. 가상 환경은 Kubernetes를 이용해 해당 운영체제 이미지를 포함한 Pod 생성, 확장, 복구를 자동화한다. 인프라는 AWS 기반으로 구축되며, EC2 인스턴스를

활용한 서버 운영, S3를 활용한 VM 상태 저장(.tut 파일), VPC를 통한 네트워크 구성, IAM을 통한 접근 제어, CloudWatch를 활용한 로그 및 자원 모니터링을 수행한다. 또한, Prometheus와 Grafana를 연계하여 자원 사용량 및 보안 상태를 시각화하고 실시간으로 감시할 수 있는 환경을 구축한다.

2) 구현 기능

플랫폼의 주요 기능으로는 사용자가 웹 UI를 통해 운영체제를 선택하고, 즉시 운영체제 이미지를 포함한 Pod 기반 가상 머신을 생성하여 실행할 수 있다. 생성된 가상 머신은 일정 시간이 경과하면 자동으로 종료되어 자원이 회수되며, WebSocket을 기반으로 한 웹 터미널을 통해 별도의 SSH 설정 없이도 브라우저 상에서 직접 조작이 가능하다. 또한, 사용자의 CPU, 메모리, 네트워크 트래픽 등의 리소스를 실시간으로 모니터링할 수 있으며, 가상 머신 상태를 저장하고 복원할 수 있는 기능도 제공된다. 관리자 대시보드를 통해 전체 가상 머신의 상태 확인, 강제 종료, 경고 메시지 발송 등의 제어가 가능하다. 이와 함께, 네트워크 접근 제한(IP 필터링) 및 사용자 접근 관리 기능도 포함되어 있어 보안성과 편의성을 모두 확보할 수 있다.

3) 시스템 구상도

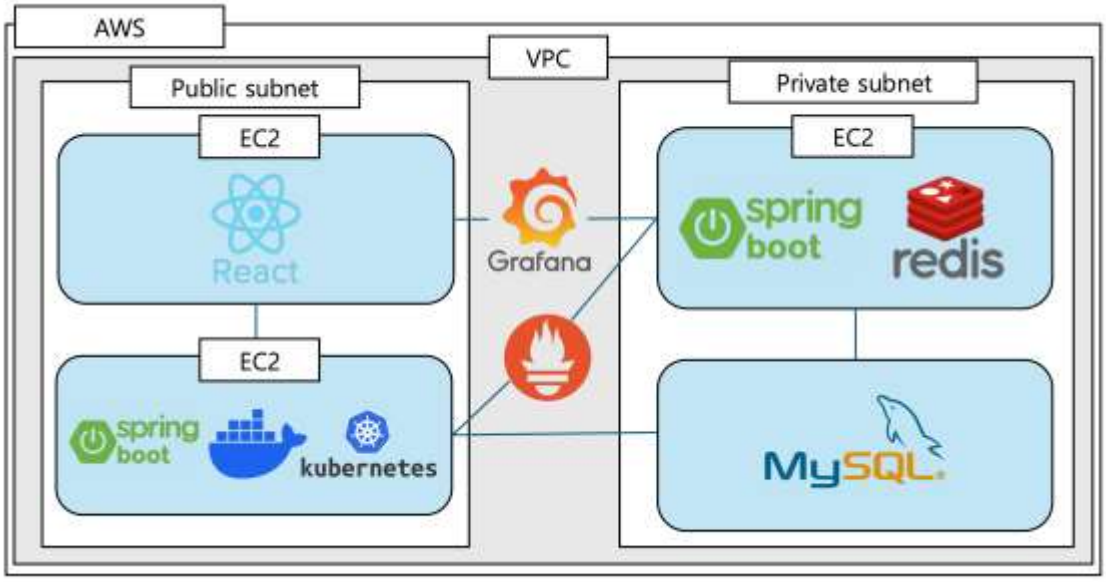


그림 1 플랫폼 아키텍처

2. 시스템 장비 요구사항

요구사항 고유번호		ECR-001
요구사항 명칭		프론트엔드 서버 구성
요구사항 분류		장비
요구사항 상세 설명	정의	사용자와 상호작용하는 웹 기반 UI를 실행할 수 있는 React 서버 구성
	세부내용	1. React 기반의 프론트엔드 서버를 AWS EC2 인스턴스에 배포하는 기능 2. 퍼블릭 서브넷에서 운영되어 사용자의 요청을 직접 처리하는 기능
산출정보		1. 웹 UI 2. 사용자 요청 로그
관련 요구사항		SFR-001, SIR-001

요구사항 고유번호		ECR-002
요구사항 명칭		백엔드 서버 구성
요구사항 분류		장비
요구사항 상세 설명	정의	사용자 요청을 처리하고 가상 머신 생성 로직을 담당하는 서버 구성
	세부내용	Spring Boot 기반 백엔드 서버로 요청을 받아 쿠버네티스에 Pod t생성을 지시하고 WebSocket 통신을 관리하는 기능
산출정보		1. 컨테이너 생성 요청 2. 로그 기록 3. WebSocket 메시지
관련 요구사항		SFR-001, SFR-003, SIR-004

요구사항 고유번호		ECR-003
요구사항 명칭		컨테이너 실행 노드 구성
요구사항 분류		장비
요구사항 상세 설명	정의	실제 가상 머신 역할을 수행하는 Pod가 실행되는 서버 노드 구성
	세부내용	1. 쿠버네티스가 설치된 EC2 인스턴스에서 사용자의 요청에 따라 운영체제를 포함한 Pod가 생성되어 실행하는 기능 2. Pod는 컨테이너 단위로 구성된 Pod를 오토스케일링 및 자동 복구를 위해 쿠버네티스 클러스터에 등록된 워커 노드에서 관리하는 기능
산출정보		1. Pod ID 2. 실행된 OS 환경 정보 3. Pod 상태 로그
관련 요구사항		SFR-001, SER-001

요구사항 고유번호		ECR-004
요구사항 명칭		모니터링 시스템 구성
요구사항 분류		장비
요구사항 상세 설명	정의	전체 시스템의 리소스 상태를 수집하고 시각화하는 모니터링 장비 구성
	세부내용	Prometheus 서버를 통해 리소스를 수집하고, Grafana 대시보드를 통해 관리자 및 사용자에게 시각적으로 제공하는 기능
산출정보		1. 실시간 리소스 사용량 2. 알림 로그
관련 요구사항		SFR-004, PER-004

요구사항 고유번호		ECR-005
요구사항 명칭		데이터베이스 및 캐시 구성
요구사항 분류		장비
요구사항 상세 설명	정의	사용자 정보 및 VM 상태 관리를 위한 데이터 저장 장치 구성
	세부내용	MySQL을 통해 사용자/이력/접속정보를 저장하고, Redis를 통해 세션 및 상태 캐싱을 처리하는 기능
산출정보		1. 사용자 계정 및 파일 복원 상태 2. 접속 이력
관련 요구사항		SFR-006, SFR-007

요구사항 고유번호		ECR-006
요구사항 명칭		VPC 및 서브넷 구성
요구사항 분류		장비
요구사항 상세 설명	정의	퍼블릭과 프라이빗 네트워크 분리를 통한 보안 및 논리적 시스템 구성
	세부내용	퍼블릭 서브넷에 프론트엔드, 모니터링 서버를 두고, 프라이빗 서브넷에는 백엔드 및 DB 서버를 구성하여 외부 노출 방지하는 기능
산출정보		1. 서브넷 IP 배치 2. 방화벽 설정 로그
관련 요구사항		SFR-007, SER-002

요구사항 고유번호		ECR-007
요구사항 명칭		S3 기반 스토리지 구성
요구사항 분류		장비
요구사항 상세 설명	정의	사용자 가상 머신 상태(.tut 파일 등) 백업 및 복원용 스토리지 구성
	세부내용	사용자가 저장을 요청한 VM 상태를 .tut 형식으로 S3에 저장하고, 이후 해당 파일을 기반으로 복원할 수 있도록 함
산출정보		1. S3 버킷 내 .tut 파일 2. 사용자 복원 이력
관련 요구사항		SFR-006, DAR-001

3. 기능 요구사항

요구사항 고유번호		SFR-001
요구사항 명칭		운영체제 선택 및 Pod 실행 기능
요구사항 분류		기능
요구사항 상세 설명	정의	사용자가 웹 UI를 통해 원하는 운영체제를 선택하고, 해당 운영체제를 실행하는 Pod를 자동으로 생성하는 기능
	세부내용	React UI에서 사용자가 Ubuntu, Kali 등의 운영체제를 선택하면, 백엔드는 쿠버네티스 API를 통해 해당 이미지를 기반으로 한 Pod를 생성하고 클러스터 내에 배포하는 기능
산출정보		1. 선택된 운영체제 정보 2. 생성된 Pod 정보
관련 요구사항		ECR-003

요구사항 고유번호		SFR-002
요구사항 명칭		가상 머신 자동 종료 기능
요구사항 분류		기능
요구사항 상세 설명	정의	일정 시간이 지나면 자동으로 실행 중인 Pod를 종료하고 리소스를 회수하는 기능
	세부내용	1. 백엔드에서 생성된 각 Pod에는 TTL(Time-To-Live) 타이머가 설정되며, 지정된 시간이 지나면 쿠버네티스 API를 통해 Pod를 자동 삭제하는 기능 2. 삭제 된 정보는 사용자에게 종료 알림으로 제공되는 기능
산출정보		1. 종료된 Pod ID 2. 종료 시각 3. 종료 로그
관련 요구사항		PER-003, ECR-003

요구사항 고유번호		SFR-003
요구사항 명칭		웹 터미널 접속 기능
요구사항 분류		기능
요구사항 상세 설명	정의	사용자가 웹 브라우저에서 생성된 가상 머신(Pod)에 직접 접속할 수 있는 터미널 기능
	세부내용	1. WebSocket을 기반으로 프론트엔드에서 xterm.js 터미널을 제공하는 기능 2. 백엔드를 통해 Pod의 컨테이너 쉘과 연결되어 명령어를 주고받는 기능
산출정보		1. 실시간 명령어 입력/출력 로그 2. 접속 시간
관련 요구사항		SIR-003, ECR-002

요구사항 고유번호		SFR-004
요구사항 명칭		상태 저장 및 복원 기능
요구사항 분류		기능
요구사항 상세 설명	정의	사용자가 작업 중이던 가상 머신 상태를 저장하고, 이후 동일한 환경으로 복원할 수 있는 기능
	세부내용	1. 사용자의 요청에 따라 현재 Pod의 상태를 확장자 파일로 변환하여 AWS S3에 저장하는 기능 2. 복원 요청 시 해당 파일을 기반으로 동일한 Pod를 재생성하는 기능
산출정보		1. 확장자 파일 2. 복원된 Pod 정보 3. 복원 로그
관련 요구사항		ECR-007, SIR-005

4. 성능 요구사항

요구사항 고유번호		PER-001
요구사항 명칭		동시 사용자 처리 성능
요구사항 분류		성능
요구사항 상세 설명	정의	다수의 사용자가 동시에 가상 머신 생성을 요청하더라도 지연 없이 응답하는 시스템 성능
	세부내용	쿠버네티스의 오토스케일링 기능과 로드밸런서를 활용하여 최소 20명의 사용자가 동시에 가상 머신을 실행해도 5초 이내에 응답이 이루어지도록 처리
산출정보		1. 응답 시간 지표 2. 활성 사용자 수 3. Pod 생성 시간 로그
관련 요구사항		SFR-001, ECR-003

요구사항 고유번호		PER-002
요구사항 명칭		Pod 생성 속도
요구사항 분류		성능
요구사항 상세 설명	정의	사용자가 운영체제를 선택한 후 Pod가 실행되기까지 걸리는 시간
	세부내용	컨테이너 이미지가 사전 빌드되어 있어야 하며, 사용자의 요청으로부터 Pod Ready 상태까지의 시간이 10초 이내여야 함
산출정보		1. Pod 생성 요청/응답 시간 2. Ready 상태 도달 시각
관련 요구사항		SFR-001, ECR-003

요구사항 고유번호		PER-003
요구사항 명칭		리소스 자동 회수 시간
요구사항 분류		성능
요구사항 상세 설명	정의	일정 시간이 지난 후 사용되지 않는 Pod를 종료하고, 리소스를 회수하는 데 걸리는 시간
	세부내용	가상 머신 사용 종료 후 최대 60초 이내에 관련 Pod를 자동 삭제하고, 메모리·CPU 등의 리소스를 재할당 가능하도록 함
산출정보		1. 종료된 Pod 로그 2. 리소스 회수 시간 측정 값
관련 요구사항		SFR-002, ECR-004

요구사항 고유번호		PER-004
요구사항 명칭		모니터링 시각화 응답 속도
요구사항 분류		성능
요구사항 상세 설명	정의	Prometheus에서 수집한 리소스 데이터를 Grafana에 실시간으로 시각화하는 데 걸리는 시간
	세부내용	CPU, 메모리, 네트워크 사용률 등의 데이터가 2초 이내에 대시보드에 반영되도록 설정하며, 사용자와 관리자 모두에게 동일한 속도로 제공되어야 함
산출정보		1. 시각화 반영 지연 시간 2. 대시보드 렌더링 로그
관련 요구사항		SFR-004, ECR-004

5. 인터페이스 요구사항

요구사항 고유번호		SIR-001
요구사항 명칭		사용자 UI 인터페이스
요구사항 분류		인터페이스
요구사항 상세 설명	정의	사용자가 가상 머신을 선택하고 실행할 수 있는 웹 기반 사용자 인터페이스
	세부내용	1. React.js로 구현되며, 운영체제 선택, VM 상태 확인, 터미널 접속 등을 지원하는 기능 2. 사용자 요청은 REST API로 백엔드에 전달하는 기능
산출정보		1. 사용자 선택 정보 2. 실행 요청 로그
관련 요구사항		SFR-001, ECR-003

요구사항 고유번호		SIR-002
요구사항 명칭		관리자 대시보드 인터페이스
요구사항 분류		인터페이스
요구사항 상세 설명	정의	리자가 시스템 전체 상태를 실시간으로 모니터링하고 제어할 수 있는 전용 인터페이스
	세부내용	1. Grafana 기반 대시보드와 관리자 전용 페이지를 제공하며, 전체 Pod 상태, 사용자 현황, 리소스 사용량 등을 시각적으로 확인할 수 있음. 2. 특정 Pod 강제 종료 기능 포함
산출정보		1. 시스템 상태 데이터 2. 제어 로그
관련 요구사항		SFR-005, PER-004, ECR-004

요구사항 고유번호		SIR-003
요구사항 명칭		WebSocket 터미널 인터페이스
요구사항 분류		인터페이스
요구사항 상세 설명	정의	사용자가 가상 머신에 실시간으로 접속할 수 있도록 지원하는 WebSocket 기반 터미널 인터페이스
	세부내용	xterm.js를 활용한 웹 터미널이 React에 포함되며, 서버를 통해 사용자의 명령어가 Pod 내부 쉘로 전달됨. 실시간 양방향 통신 보장
산출정보		1. 터미널 입력/출력 로그 2. 접속 이력
관련 요구사항		SFR-003, ECR-002

6. 데이터 요구사항

요구사항 고유번호		DAR-001
요구사항 명칭		가상 머신 상태 저장 데이터
요구사항 분류		데이터
요구사항 상세 설명	정의	사용자의 가상 머신 환경을 저장하고 복원할 수 있도록 상태 데이터를 관리하는 요구사항
	세부내용	1. 사용자의 요청 시 VM의 현재 상태를 확장자 파일 형태로 변환하고, AWS S3에 저장 2. 저장한 파일은 Pod 재생성 시 동일한 환경으로 복원하는 데 사용
산출정보		1. 확장자 파일 2. S3 저장 경로 3. 사용자 ID와의 매핑 정보
관련 요구사항		SFR-004, ECR-007

요구사항 고유번호		DAR-002
요구사항 명칭		리소스 사용량 데이터
요구사항 분류		데이터
요구사항 상세 설명	정의	가상 머신(Pod) 실행 중 발생하는 CPU, 메모리, 네트워크 등의 사용 데이터를 수집하고 저장하는 요구사항
	세부내용	1. Prometheus를 통해 주기적으로 Pod 리소스 사용량을 수집하며, Grafana를 통해 시각화 2. 데이터는 일정 기간 동안 저장되어 추후 성능 분석 및 경고 알림에 활용
산출정보		1. CPU % 2. RAM MB 3. 네트워크 전송량 등 리소스 로그
관련 요구사항		PER-004, SFR-005, ECR-004

요구사항 고유번호		DAR-003
요구사항 명칭		사용자 접속 및 사용 이력 데이터
요구사항 분류		데이터
요구사항 상세 설명	정의	사용자의 가상 머신 생성/접속/종료 이력 및 터미널 사용 기록 등을 기록하고 조회할 수 있는 요구사항
	세부내용	MySQL DB에 사용자 ID, 접속 IP, 사용한 운영체제, 시작 시각, 종료 시각, 명령어 기록 등 다양한 사용 이력을 저장하며, 추후 관리자 페이지에서 확인 가능
산출정보		1. 사용자 이력 로그 2. 접속 기록 3. 명령어 입력 내역
관련 요구사항		SFR-003, SIR-002, SIR-003

7. 테스트 요구사항

요구사항 고유번호		TER-001
요구사항 명칭		가상 머신 실행 및 접속 테스트
요구사항 분류		테스트
요구사항 상세 설명	정의	사용자가 선택한 운영체제가 정상적으로 실행되고, 웹 터미널로 접속 가능한지 검증하는 테스트 요구사항
	세부내용	각 운영체제(Ubuntu, Kali 등)에 대해 Pod가 정상적으로 생성되고, 웹 터미널에서 로그인 및 명령어 입력이 정상 동작하는지 테스트
산출정보		1. Pod 상태 로그 2. WebSocket 연결 로그 3. 테스트 결과 리포트
관련 요구사항		SFR-001, SFR-003, SIR-003

요구사항 고유번호		TER-002
요구사항 명칭		동시 사용자 부하 테스트
요구사항 분류		테스트
요구사항 상세 설명	정의	여러 사용자가 동시에 가상 머신을 요청했을 때, 시스템이 성능 저하 없이 응답하는지 확인하는 테스트 요구사항
	세부내용	10명, 20명, 30명의 사용자 시나리오를 구성하여 동시에 운영체제를 실행하고, 응답 시간 및 시스템 부하 데이터를 측정
산출정보		1. 사용자 수 대비 응답 시간 2. 자원 사용률 3. 실패 요청 수
관련 요구사항		PER-001, PER-002, ECR-003

요구사항 고유번호		TER-003
요구사항 명칭		VM 상태 저장 및 복원 테스트
요구사항 분류		테스트
요구사항 상세 설명	정의	가상 머신 상태 저장 후 확장자 파일을 통해 동일한 환경으로 복원 가능한지 확인하는 테스트 요구사항
	세부내용	사용자 작업 내역(파일 생성, 패키지 설치 등)을 포함한 상태를 저장하고, 복원 후 동일한 작업 환경이 재현되는지 검증
산출정보		1. 확장자 파일 2. 복원 후 상태 비교 결과
관련 요구사항		SFR-004, DAR-001, ECR-007

8. 보안 요구사항

요구사항 고유번호		SER-001
요구사항 명칭		IP 기반 접근 제어 기능
요구사항 분류		보안
요구사항 상세 설명	정의	가상 머신(Pod)에 접속할 수 있는 사용자의 IP를 제한하여 불필요한 외부 접근을 차단하는 기능
	세부내용	1. 사용자는 VM 생성 시 자신의 접속 IP만 허용하도록 설정할 수 있으며, 관리자도 허용 IP 목록을 설정. 2. 쿠버네티스 네트워크 정책 또는 방화벽 설정으로 적용
산출정보		1. 허용된 IP 리스트 2. 접근 차단 로그
관련 요구사항		SIR-003, ECR-006

요구사항 고유번호		SER-002
요구사항 명칭		포트 접근 제한 정책
요구사항 분류		보안
요구사항 상세 설명	정의	외부에 노출되는 포트를 최소화하여 서비스 노출 범위를 제한하는 기능
	세부내용	Pod는 사용 목적에 따라 특정 포트만 오픈되며, 나머지 포트는 방화벽 및 보안 그룹 설정으로 차단됨
산출정보		1. 방화벽 설정 2. 오픈 포트 목록
관련 요구사항		ECR-006, SFR-003

9. 품질 요구사항

요구사항 고유번호		QAR-001
요구사항 명칭		시스템 신뢰성
요구사항 분류		품질
요구사항 상세 설명	정의	시스템이 예기치 않은 오류 없이 안정적으로 동작해야 함
	세부내용	Pod 생성/종료, 터미널 접속 등 주요 기능은 실패율이 1% 미만이어야 하며, 장애 발생 시 자동 복구 절차가 실행되어야 함
산출정보		1. 오류 발생률 2. 장애 대응 로그

요구사항 고유번호		QAR-002
요구사항 명칭		유지보수 용이성
요구사항 분류		품질
요구사항 상세 설명	정의	시스템은 향후 기능 추가 또는 수정 시에도 쉽게 유지보수 가능해야 함
	세부내용	프론트엔드와 백엔드는 모듈화되어 있고, 컨테이너 단위로 구성되어 있어 서비스 중단 없이 배포 및 롤백 가능해야 함
산출정보		1. 배포 로그 2. rollback 이력

요구사항 고유번호		QAR-003
요구사항 명칭		확장 가능성
요구사항 분류		품질
요구사항 상세 설명	정의	사용자 증가 또는 기능 확장 시 시스템 자원이 유연하게 대응 가능해야 함
	세부내용	쿠버네티스의 오토스케일링 기능을 활용하여 트래픽이 급증하더라도 Pod를 자동으로 확장할 수 있어야 함
산출정보		1. 자동 스케일링 이력 2. 사용자 수 대비 자원 증감 기록

요구사항 고유번호		QAR-004
요구사항 명칭		사용자 편의성(직관성)
요구사항 분류		품질
요구사항 상세 설명	정의	사용자는 복잡한 절차 없이 서비스를 사용할 수 있어야 함
	세부내용	운영체제 선택 → 실행 → 접속까지 3단계 이하로 구성되며, 별도 가이드 없이 사용 가능한 UI 제공
산출정보		1. UX 테스트 결과 2. 사용자 피드백

10. 제약사항

요구사항 고유번호	CR-001		
요구사항 명칭	AWS 기반 인프라 사용 제한		
요구사항 분류	제약사항	응락수준	필수
요구사항 세부내용	시스템은 AWS 인프라(EC2, S3, VPC 등)를 기반으로만 구축되어야 하며, 타 클라우드 환경(GCP, Azure 등)은 사용하지 않음		

요구사항 고유번호	CR-002		
요구사항 명칭	예산 및 리소스 상한 제한		
요구사항 분류	제약사항	응락수준	권장
요구사항 세부내용	EC2 인스턴스, S3 저장 용량 등 클라우드 리소스는 과금 및 무상 크레딧 범위를 초과하지 않도록 설정하며, 트래픽 자동 확장은 불가함		

11. 프로젝트 관리 요구사항

요구사항 고유번호	PMR-001		
요구사항 명칭	버전관리		
요구사항 분류	프로젝트 관리	응락수준	선택
요구사항 세부내용	- 소스코드 및 구성 파일의 버전 관리 기능 - 다중 브랜치 병합 기능을 통한 여러 개발자 동시 작업 가능 - 팀원 간의 협업을 위한 기능 포함으로 코드 공유 및 협업 작업에 용이		

요구사항 고유번호	PMR-002		
요구사항 명칭	팀 협업 도구		
요구사항 분류	프로젝트 관리	응락수준	선택
요구사항 세부내용	- 프로젝트 관리를 위한 팀 협업 도구 - Notion을 활용한 팀 협업 플랫폼 사용 - 팀원 간의 업무 할당 및 작업 상태 공유를 위한 알림		

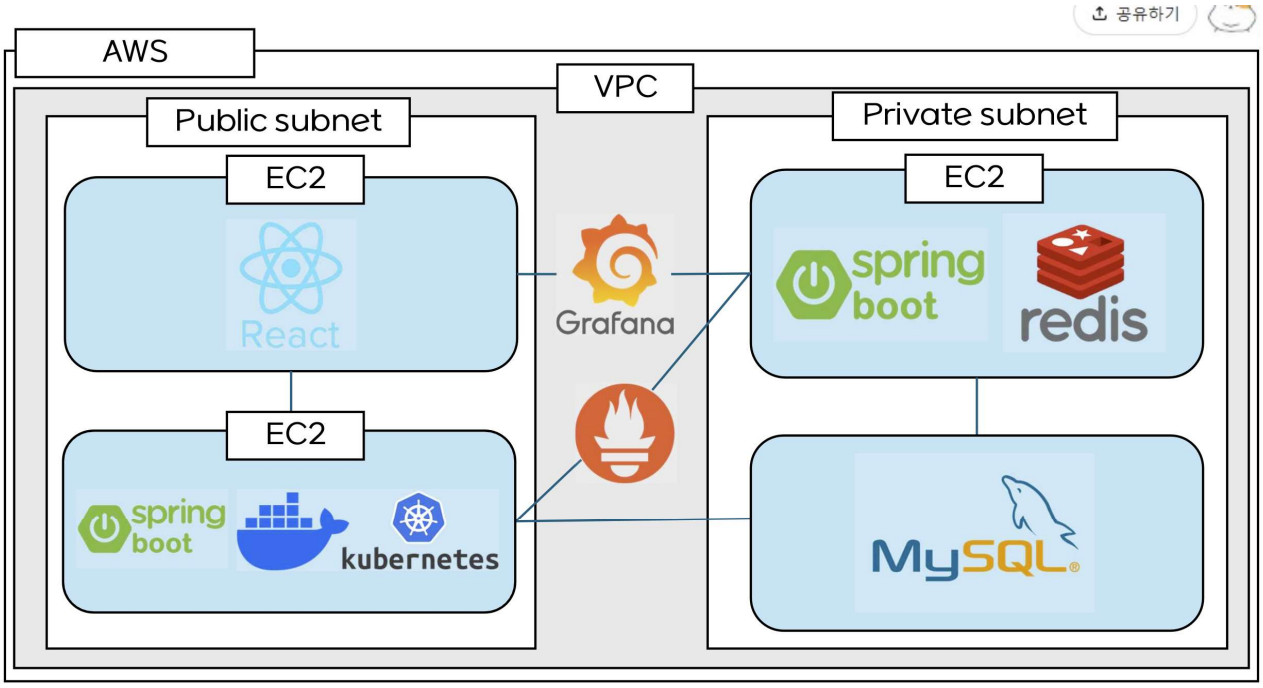
요구사항 고유번호	PMR-003		
요구사항 명칭	작업 일정 관리		
요구사항 분류	프로젝트 관리	응락수준	선택
요구사항 세부내용	- 프로젝트의 작업 일정 관리 및 추적 기능 - 진행상황 실시간 업데이트를 통한 효율적인 프로젝트 관리		

[별첨2]

중간보고서

1. 요구사항 정의서에 명세된 기능에 대하여 현재까지 분석, 설계, 구현(소스코드 작성) 및 테스팅한 내용을 기술하시오.

전체 시스템 구상도



현재 동일한 공유기(로컬 네트워크) 상에서 다음과 같은 구성으로 개발 진행 중에 있다.

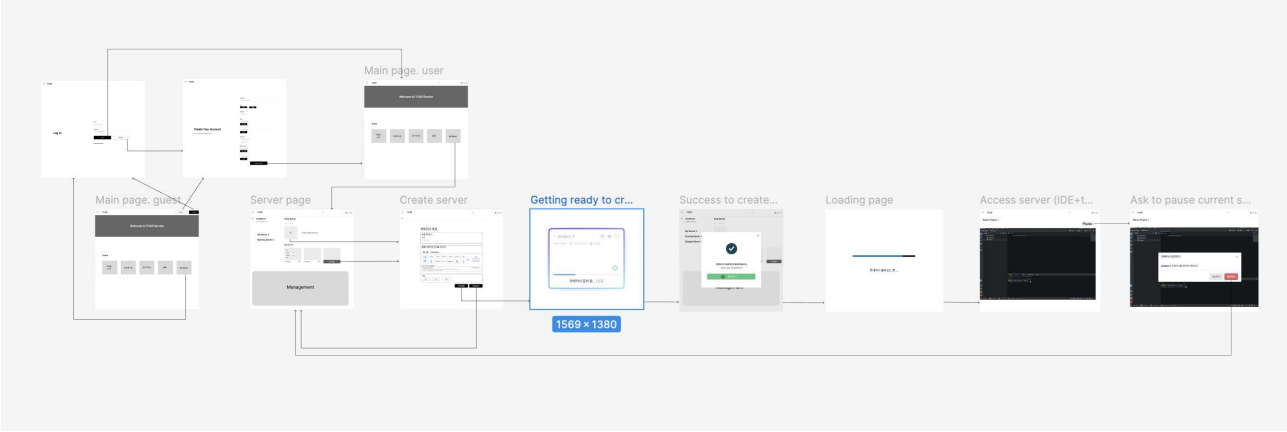
- 프론트엔드 : 192.168.1.9 (React 기반 Web UI)
- 백엔드 : 192.168.1.19 (Spring Boot 기반 인증, 회원 관리, 시스템 관리, 중계)
- 가상머신 서버 : 192.168,1,23 (Kubernetes Pod 제어를 위한 서버, Spring Boot 기반)

개발환경에서는 Kubernetes를 가상 머신 서버 내부에서 minikube를 이용해 구동 중이며, minikube ip는 10.211.55.6 IP 내역을 사용하여 개발 진행 중에 있다.

□ 프론트엔드

1. 와이어프레임 작성

기초 기능 : 메인화면, 로그인, 회원가입, 유저 정보 수정
핵심 기능 : 개인 서버 생성, 개인 서버 삭제, 웹 터미널 접속



2. 회원가입, 로그인 코드 구현

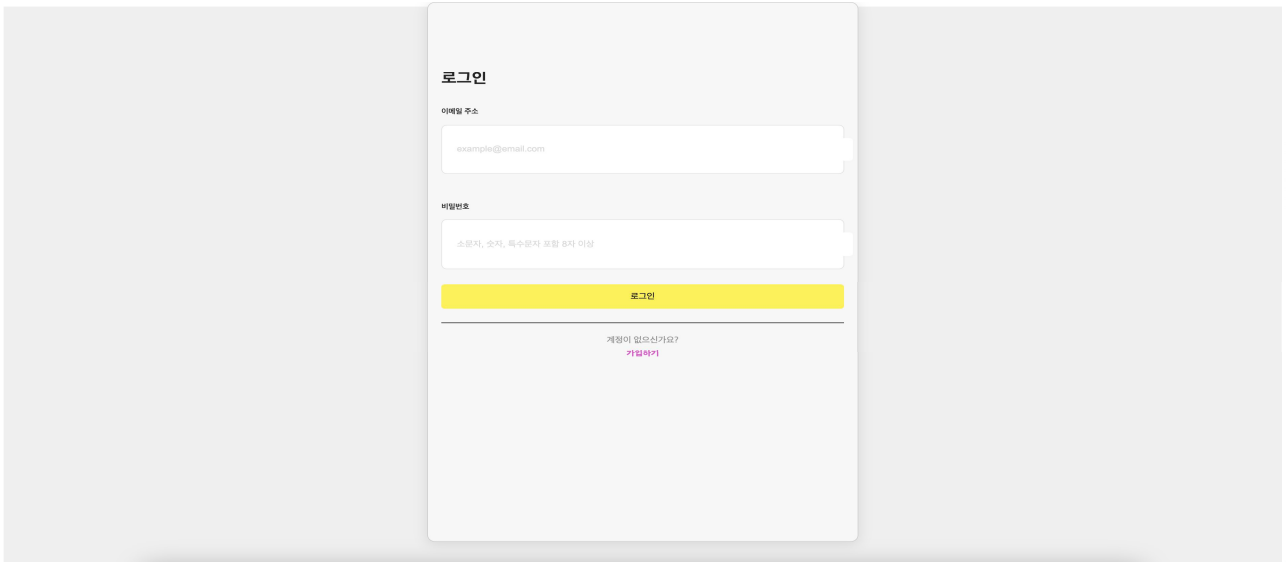


그림 5 로그인 UI

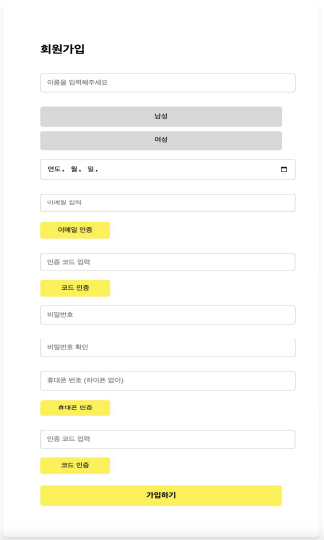


그림 6 회원가입 UI

이때, 사용자의 이메일은 로그인 아이디로 사용되기에, 1개의 이메일은 1명의 사용자에게만 할당되도록 설계하였다. 또한 사용자가 여러 개의 계정을 중복 생성하지 못하도록 하기 위해, 휴대폰 인증을 통해 중복 방지 정책을 적용하였다. 즉, 하나의 휴대폰 번호로는 하나의 계정만 인증 및 생성할 수 있도록 제한을 두어 보안성과 사용자 관리의 신뢰도를 높였다.

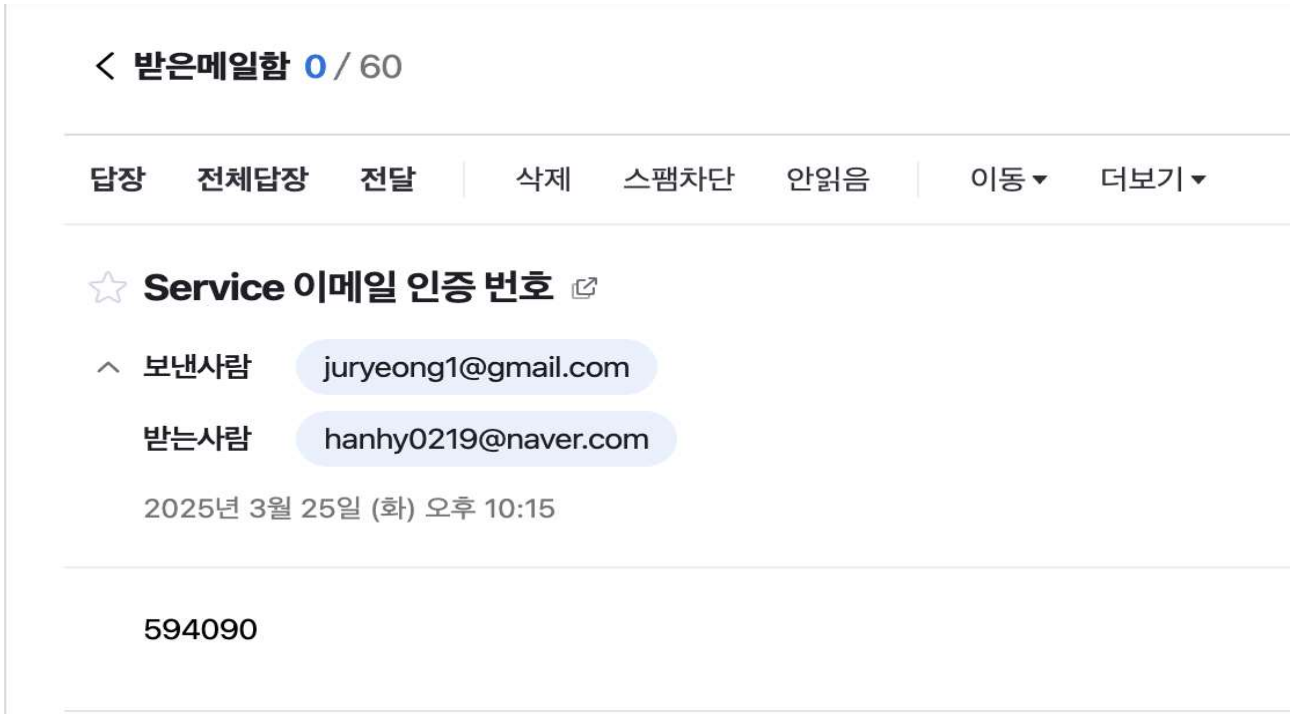


그림 7 이메일 인증을 통한 보안 강화

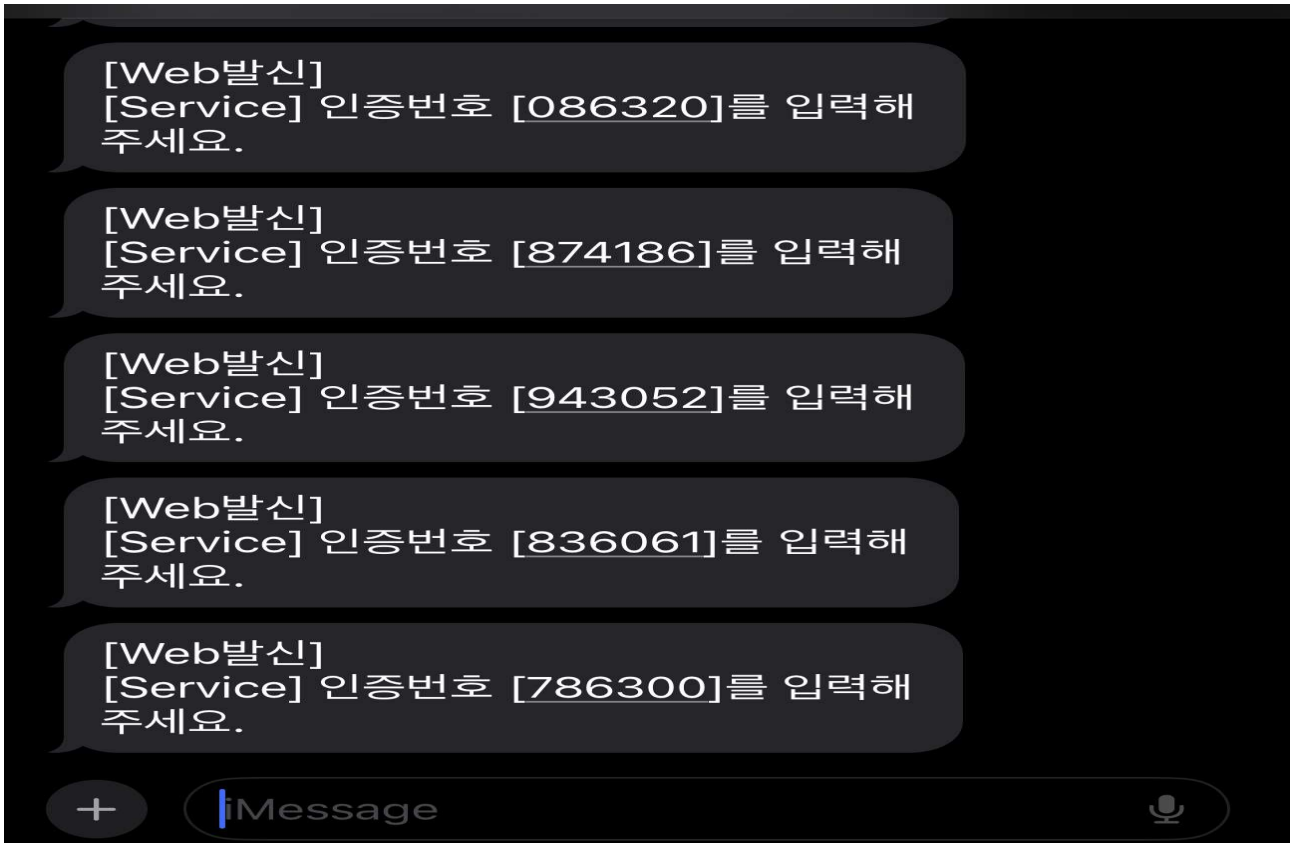


그림 8 핸드폰 인증을 통한 보안 강화

3. 메인페이지, 서버 생성 페이지 구현

백엔드로부터 생성가능한 서버 목록들을 가져와서 사용자로부터 서버 생성을 요청하면, 서버가 생성되는 과정까지의 화면 페이지 및 실제 테스트까지 완료되었다.

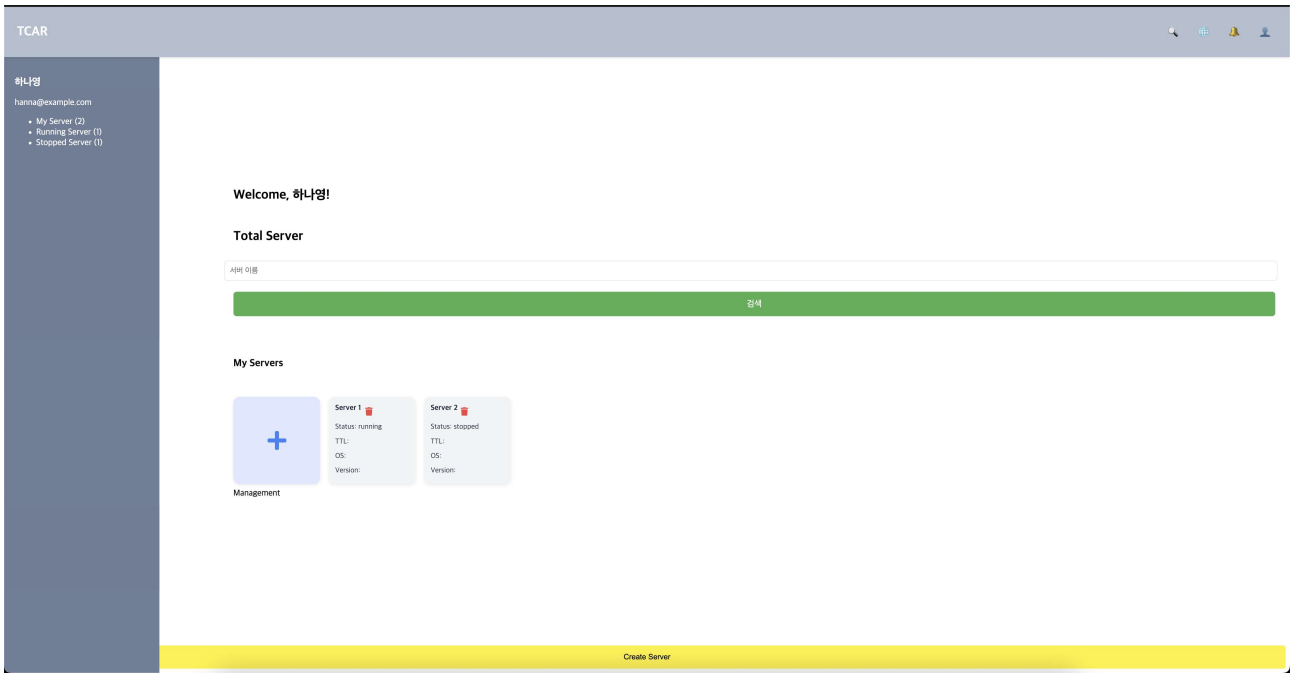


그림 9 메인페이지(임시)

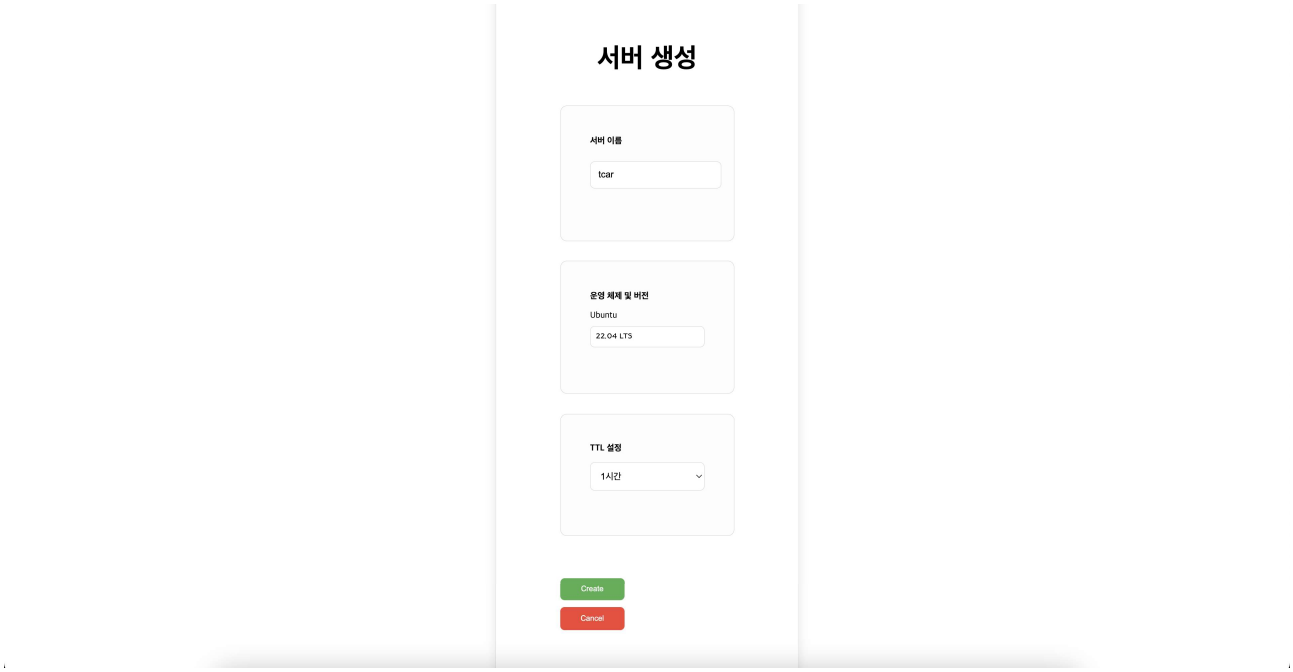


그림 10 서버생성 페이지(임시)

□ 백엔드

1. 회원가입 및 JWT 토큰, REDIS를 이용한 로그인

Java 복사 캡션 ···

```java
package com.hanbat.tcar.common;

@Component
public class JwtGenerator {
    private final Key SECRET_KEY;
    private final long ACCESS_TOKEN_EXPIRATION = 1000 * 60 * 15; // 15분
    private final long REFRESH_TOKEN_EXPIRATION = 1000L * 60 * 60 * 24 * 7; /

    public JwtGenerator(@Value("${jwt.secret.key}") String secretKey) {
        byte[] keyBytes = Decoders.BASE64.decode(secretKey);
        this.SECRET_KEY = Keys.hmacShaKeyFor(keyBytes);
    }

    public JWToken generateToken(User user){
        Date now = new Date();
        Date accessTokenExpiry = new Date(now.getTime() + ACCESS_TOKEN_EXPIRA
        Date refreshTokenExpiry = new Date(now.getTime() + REFRESH_TOKEN_EXPI

        // 액세스 토큰 생성
        String accessToken = Jwts.builder()
                .subject(user.getEmail())
                .claim("userId", user.getId())
                .claim("userName", user.getUsername())
                .issuedAt(now)
                .expiration(accessTokenExpiry)
                .signWith(SECRET_KEY, SignatureAlgorithm.HS256)
                .compact();

        // 2) Refresh Token
        String refreshToken = Jwts.builder()
                .subject(user.getEmail())
                .claim("userId", user.getId())
                .claim("userName", user.getUsername())
                .issuedAt(now)
                .expiration(refreshTokenExpiry)
                .signWith(SECRET_KEY, SignatureAlgorithm.HS256)
                .compact();

        return JWToken.builder()
                .accessToken(accessToken)
                .refreshToken(refreshToken)
                .build();
    }
```

그림 11 JWT토큰 발급(액세스 토큰, 리프레시 토큰) 코드

```java
package com.hanbat.tcar.common;

@Component
public class JwtProvider {
    private final SecretKey SECRET_KEY;

    public JwtProvider(@Value("${jwt.secret.key}") String secretKey) {
        byte[] keyBytes = Decoders.BASE64.decode(secretKey);
        this.SECRET_KEY = Keys.hmacShaKeyFor(keyBytes); // SecretKey로 저장
    }

    // JWT 검증 메서드 - 토큰 유효성 검증
    public boolean validateToken(String token) {
        try {
            Jwts.parser()
                    .verifyWith(SECRET_KEY)
                    .build()
                    .parseSignedClaims(token);

            return true;  // 유효한 토큰
        } catch (ExpiredJwtException e) {
            System.out.println("JWT 만료됨");
        } catch (MalformedJwtException e) {
            System.out.println("JWT 형식 오류");
        } catch (Exception e) {
            System.out.println("JWT 검증 실패: " + e.getMessage());
        }
        return false;
    }

    // 토큰에서 이메일(사용자 정보 - subject) 추출
    public String getEmailFromToken(String token) {
        return Jwts.parser()
                .verifyWith(SECRET_KEY)  // `setSigningKey()` 대신 사용
                .build()
                .parseSignedClaims(token)
                .getPayload()
                .getSubject();
    }
}
```

그림 12 JWT 발급 및 검증

HTTP는 비연결성 특성을 가지기에, 사용자 인증 상태를 유지하기 위해 쿠키나 세션 기반의 인증이 사용되어 왔으나, 보안성이 떨어지기에, 많은 곳에서 사용하는 JWT(Json Web Token)을 인증방식을 채택하여 로그인을 구현하였다. 개발환경에서는 쿠키 방식의 토큰 전달이 HttpOnly, Secure, Cors 등의 제약으로 인해 어려움이 있기에, JWT 는 Authorization 헤더의 BEARER 방식으로 전달되며, 토큰은 프론트엔드의 Local Storage에 일시적으로 저장되어 인증시에만 사용하도록 설계하였다. 또한 JWT 만료 시간 및 토큰 유효성 관리를 위해 발급된 토큰 정보를 Redis에 저장하고, 만료 시간이 지나면 자동으로 삭제되도록 하여 보안성과 자원 관리의 효율성을 높였다.

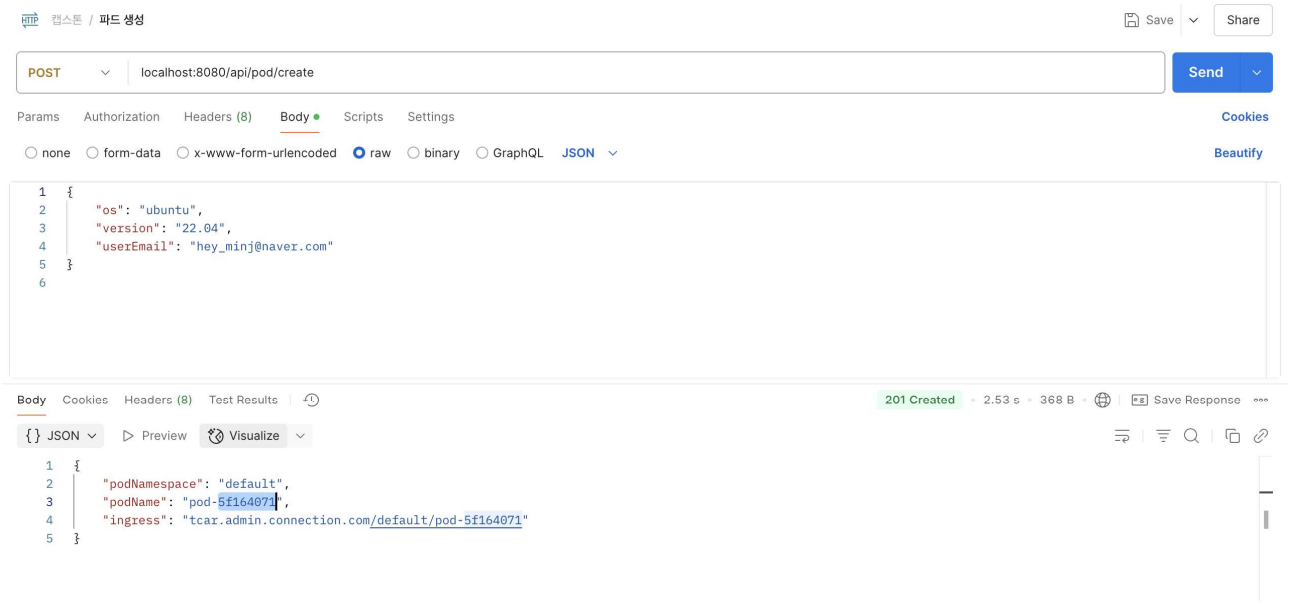
3. 가상머신 서버와의 중계

회원의 로그인 인증, 사용자 검증을 하는 코드들은 백엔드에서 모든 것을 담당하기에 가상 머신의 보안성을 증진시키기 위하여 백엔드가 사용자들을 검증하고, 검증이 완료된 경우 가상 머신 서버로 요청을 보내는 방식으로 아키텍쳐를 구성하였다. 추후 nonce 값을 보내서 백엔드에서 검증된 안전한 정보라는 것을 보태어 넣거나, 가상머신 서버에서 IP허용을 백엔드로부터 오는 IP만 허용하도록 설정하는 화이트리스트 방식을 적용하여 보안성을 높일 예정이다. 가상 머신 서버와의 서버 생성, 삭제, 접속 기능은 완료된 상태이며, 프론트로부터 오는 요청까지 추후 테스트할 예정이다.

□ 가상머신 서버

초기에는 docker-java를 이용하여 ubuntu 22.04 기반 컨테이너를 생성하였으며, 이후에는 kubernetes API를 직접 사용하여 Pod를 생성하는 방식으로 전환하였다. 두 방식 모두 기능 구현과 테스트를 마쳤으며, 현재는 Kubernetes 기반 Pod 생성 방식으로 최종 개발 방향을 설정하였다.

1. Pod 생성



2. Pod 삭제

캡스톤 / 파드 삭제

Save | Share

DELETE | localhost:8080/api/pod/delete | Send

Params | Authorization | Headers (8) | Body | Scripts | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON | Beautify

```
{
    "userEmail" : "hey_minj@naver.com",
    "podNamespace" : "default",
    "podName" : "pod-d9a7439c"
}
```

Body | Cookies | Headers (8) | Test Results | 200 OK | 470 ms | 281 B | Save Response

{} JSON | Preview | Visualize

```
{
    "message": "삭제 성공!"
}
```

(3) 접근 검증

캡스톤 / 접근 검증

Save | Share

GET | http://localhost:8080/api/access/presigned/validate?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJoZXIfbWluakBuYXZlci5jb20iLCJwb2ROYW1lIjoicG9kLTQ0NGJIMI... | Send

Params | Authorization | Headers (6) | Body | Scripts | Settings | Cookies

Query Params

Key	Value	Description	Bulk Edit
token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJoZXIfbWluak...		
Key	Value	Description	

(4) 웹 터미널 접속

웹 터미널 테스트

🔧 사용자의 가상머신 CLI

```
[+] Connected to your pod
[+] 터미널 접속됨
root@pod-5f164071:/# echo ""

root@pod-5f164071:/# echo "hi"
hi
root@pod-5f164071:/# ls
bin   dev   home  media  opt   root  sbin  sys  usr
boot  etc   lib   mnt    proc  run   srv   tmp  var
root@pod-5f164071:/# echo

root@pod-5f164071:/#
```

DevTools is now available in Korean!

Always match Chrome's language | Switch DevTools to Korean | Don't show again

Elements | Console | Sources | Network | Performance | Memory | Application

Styles | Computed | Layout | Event Listeners

Console | Autofill | Issues

```
RECV: c
SEND: "h"
RECV: h
SEND: "o"
RECV: o
SEND: "\r"
RECV:
[?2004l
RECV:
[?2004h ]0;root@pod-5f164071: / root@pod-5f164071:/#
```

(5) 쿠버네티스 운영

```
seo ~ k get all
NAME                 READY   STATUS    RESTARTS   AGE
pod/pod-29f239f1     1/1     Running   0          25h
pod/pod-5f164071     1/1     Running   0          24h

NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/admin-service         ClusterIP   10.101.215.89    <none>        80/TCP     40d
service/basic-service         ClusterIP   10.107.114.113   <none>        80/TCP     40d
service/kubernetes            ClusterIP   10.96.0.1        <none>        443/TCP    44d
service/svc-pod-29f239f1      ClusterIP   10.100.114.115   <none>        80/TCP     25h
service/svc-pod-5f164071      ClusterIP   10.105.154.252   <none>        80/TCP     24h
seo ~
```

(6) PreSignedURL 검증

```java
@Service
public class URLValidateService {
    private final SecretKey SECRET_KEY;


    public URLValidateService(PreSignedUrlConfig preSignedUrlConfig){
        byte[] keyBytes = Decoders.BASE64.decode(preSignedUrlConfig.getSecretKey(
        this.SECRET_KEY = Keys.hmacShaKeyFor(keyBytes);
    }


    //*** TOKEN 검증 및 payload return
    public Map<String, String> getClaimsFromToken(String token){
        try{
            Claims claims = Jwts.parser()
                    .verifyWith(SECRET_KEY)
                    .build()
                    .parseSignedClaims(token)
                    .getPayload();

            return Map.of(
                    "containerId", claims.get("containerId", String.class),
                    "port", claims.get("port", String.class),
                    "userEmail", claims.get("userEmail", String.class)
            );
        } catch (ExpiredJwtException e){
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "토큰이 만료도
        } catch (MalformedJwtException e){
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "유효하지 않은
        } catch (Exception e){
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "토큰 검증에
        }

    }


}
```

소스코드 :
https://github.com/Vak-kas/TCAR_VM/tree/feature/terminal
http://github.com/TH-Carry-and-Run/frontend
https://github.com/kikijuju/backend

명세서 및 시나리오 및 설명 - 노션 :
https://www.notion.so/TCAR-172ea1a2bab480039688c3b19d2966d2

2. 프로젝트 수행을 위해 적용된 추진전략, 수행 방법의 결과를 작성하고, 만일 적용과정에서 문제점이 도출되었다면 그 문제를 분석하고 해결방안을 기술하시오.

(1) 문제 해결 전략 및 적용 방법

프로젝트 초기에는 모든 사용자 요청을 백엔드를 거쳐 가상머신 서버로 중계하는 방식으로 꾸조를 설계하여, 인증 및 보안성을 강화하고자 하였다. 그러나 이 구조는 실시간성 저하 및 백엔드 트래픽 과부하 문제를 유발할 수 있음을 확인하였다. 이에 따라 구조를 개선하여 사용자가 본인의 가상머신/운영체제 서버(Pod)에 접속할 경우, 백엔드를 거치지 않고 직접 가상머신 서버로 연결하도록 아키텍처를 전환하였다.

개선 전략 :
사용자가 본인의 가상머신(Pod)에 접속할 경우, 백엔드를 거치지 않고, 가상머신 서버에 직접 접속하도록 구조 변경.

보안 문제 해결 :
접속 시 사용자의 인증 정보를 백엔드가 검증하는 것까지는 동일. 이때 백엔드는 가상머신 서버로 리다이렉트를 시키거나 가상머신 서버로 요청을 보내는 것이 아닌, 사용자가 직접 가상머신 서버로 접속할 수 있는 JWT를 통한 PreSignedUrl 링크 생성. 가상머신서버에서는 해당 PreSigendURL을 검증.

접속 방식 전환 :
인증에 성공한 경우, 가상 머신 서버는 PreSignedURL을 검증하고, 사용자는 이를 안전하게 WebSocket으로 Pod에 접속 가능하도록 구성. 발급 기관(백엔드), 검증기관(가상머신 서버)가 독립적으로 작동하기에 각각의 부담을 줄일 수 있음.

(2) 팀원 역할 및 수행

팀원	주요역할	수행 결과
김주령	백엔드 아키텍처 설계, 인증 흐름 설계	PreSigned URL 생성 및 인증 로직 구현, 백엔드 구조 간결화
서민재	가상머신 서버 개발, 터미널 연결 기능 구현	Kubernetes Pod 접속, WebSocket구현, JWT검증 로직 개발
한하영	UX/UI 설계 및 시스템 기획	와이어 프레임 제작, 시나리오 기획 및 페이지 흐름 설계

팀원 간 역할이 명확히 분배되어 있어 병렬 개발이 원활하게 이루어졌으며, 하나의 동일 네트워크에서 테스트 환경을 구성하여 기능 간 연동 오류를 효과적으로 점검 가능하도록 하였다.

(3) 프로젝트 일정 계획에 맞추지 못한 경우의 문제점, 해결 방안

프로젝트 진행 중 팀원의 건강 문제로 인해 정기 회의 및 개발이 어려운 상황이 발생하였고, 이에 다라 초기 기획-설계-개발의 순차적 방식 대신, 개발과 기획을 병행하며 점진적으로 요구사항을 보완해 나가는 애자일 방식으로 전략을 변경하였다. 초기에는 Docker-java 기반의 컨테이너 생성 방식을 사용하였으나, 개발을 진행하면서 확장성, 관리 편의성, 모니터링 기능 등을 고려해 Kubernetes 기반 아키텍처로 변경하였다. 이 과정에서 95% 이상의 코드를 전면 수정해야 했지만, 전체 구조를 다시 정비하고, 코드 최적화 하였기에 더 안정적이고 유지보수가 쉬운 구조를 갖출 수 있었다.

(4) 요구사항 변경 관리의 결과, 문제점, 해결방안

Docker 기반에서 kubernetes 기반의 전환은 단순한 기술 교체가 아닌, 아키텍처 전면 재설계가 필요한 작업이었다. 초기에는 Docker 환경에서도 지금까지 나온 요구사항을 토대로 원하는 기능을 구현할 수 있다고 판단하였으나, 관련 기술에 대한 이해 부족으로 인해 제약을 인지학지 못했고, 이후 학습을 통해 한계를 파악한 후 구조를 변경하게 되었다. 이로 인해 많은 코드가 폐기되었으나, 이전 코드의 한계를 명확히 분석하고 잘못된 흐름을 개선하는 계기가 되었고, 불필요하게 복잡했던 책임 분산 구조를 정리할 수 있었다. 결과적으로 구조가 간결해지고, 유지보수성과 확장성이 대폭 향상되었으며, 향후 요구사항 변경에도 유연하게 대응할 수 있는 기반이 마련되었다.