Public Cloud를 활용한 Problem Soving
Football Ticketing System

Public Cloud : Google Cloud Platform (GCP)

소속	영남대학교 기계IT대학 컴퓨터공학과
학번	22213489
이름	표주원

목차 (Table of Contents)

1. 문제 정의 및 배경 (Problem Definition and Background)
    1.1 문제 상황 (Challenges in Current Ticketing Systems)
    1.2 해결 목표 (Project Goals using GCP)
    1.3 기대 효과 (Expected Benefits)

2. 시스템 아키텍처 설계 (System Architecture Design)
    2.1 전체 아키텍처 개요 (Overall Architecture Overview)
    2.2 주요 구성 요소 (Key Components Detailed Analysis)
        2.2.1 프론트엔드 계층 (Presentation Layer)
        2.2.2 애플리케이션 계층 (Application Layer)
        2.2.3 데이터베이스 계층 (Database Layer)
    2.3 네트워크 아키텍처 (Network Architecture)
        2.3.1 VPC (Virtual Private Cloud) 구성
        2.3.2 로드밸런서 구성
        2.3.3 보안 그룹 설정 (Firewall Rules)
    2.4 데이터베이스 설계 (Database Design)
        2.4.1 ERD (Entity Relationship Diagram)
        2.4.2 테이블 상세 설계 (Detailed Table Schema)
        2.4.3 인덱스 전략 (Indexing Strategy for Performance)

3. Public Cloud 구현 상세 (Public Cloud Implementation Details)
    3.1 자원 관리 (Resource Management)
        3.1.1 Compute Engine 인스턴스 구성 (Instance Configuration)
        3.1.2 Managed Instance Group (MIG) 구성 (Group Configuration)
        3.1.3 오토스케일링 동작 원리 (Operational Dynamics)
    3.2 네트워크 관리 (Network Management)
        3.2.1 로드밸런서 상세 구성 (Load Balancer Detailed Configuration)
        3.2.2 트래픽 분산 알고리즘 (Traffic Distribution Algorithm)
        3.2.3 서브넷 구성 (Subnet Configuration)
    3.3 보안 설정 (Security Configuration)
        3.3.1 사용자 권한 관리 (IAM - Identity and Access Management)
        3.3.2 네트워크 보안 (Network Security - Firewall Rules)
    3.4 데이터베이스 활용 (Database Utilization)
        3.4.1 Cloud SQL 구성 (Cloud SQL Configuration)
        3.4.2 데이터베이스 최적화 (Database Optimization)
        3.4.3 데이터베이스 모니터링 (Database Monitoring)

4. 부하 테스트 설계 및 준비 (Load Testing Design and Preparation)

4.1 부하 테스트 목적 및 범위 (Testing Objectives and Scope)
4.1.1 테스트 목표 정의
4.1.2 테스트 시나리오 설계
4.1.3 성공 기준 정의 (Success Criteria)
4.2 테스트 환경 구성 (Test Environment Setup)
4.2.1 초기 시스템 상태
4.2.2 오토스케일링 설정 확인
4.2.3 모니터링 도구 설정
4.3 부하 생성 메커니즘 (Load Generation Mechanism)
4.3.1 부하 생성 방식
4.3.2 트래픽 패턴

5. 부하 테스트 결과 (Load Test Results)
5.1 초기 상태 측정 (Baseline Measurement)
5.1.1 시스템 초기 상태 (Initial System Status)
5.1.2 초기 성능 지표 (Initial Performance Metrics)
5.2 부하 테스트 시작 (Load Test Initiation)
5.2.1 로그인 페이지 부하 (Login/Signup Load Scenario)
5.2.2 CPU 사용률 증가 (CPU Utilization Rise)
5.2.3 부하 테스트 통계 (Load Test Statistics)
5.3 오토스케일링 확장 과정 (Auto-Scaling Scale-Up Process)
5.3.1 1차 확장: 2→3개 (First Scale-Up: 2→3 Instances)
5.3.2 2차 확장: 3→4개 (Second Scale-Up: 3→4 Instances)
5.3.3 3차 확장: 4→5개 (Third Scale-Up: 4→5 Instances)
5.3.4 확장 타이밍 분석 (Scale-Up Timing Analysis)
5.4 로드밸런서 성능 분석 (Load Balancer Performance Analysis)
5.4.1 트래픽 분산 분석 (Traffic Distribution Analysis)
5.4.2 백엔드 상태 모니터링 (Backend Health Monitoring)
5.4.3 응답 시간 분석 (Response Time Analysis)
5.5 부하 중지 및 스케일 다운 (Load Stop & Scale-Down Process)
5.5.1 부하 테스트 종료 (Load Termination)
5.5.2 CPU 사용률 감소 (CPU Reduction Trend)
5.5.3 스케일 다운 과정 (Scale-Down Process)
5.5.4 축소 타이밍 분석 (Scale-Down Timing Analysis)

6. 시스템 성능 분석 (System Performance Analysis)
6.1 오토스케일링 효율성 분석 (Auto-Scaling Efficiency Evaluation)
6.1.1 확장 속도 평가 (Scale-Up Speed Assessment)
6.1.2 축소 속도 평가 (Scale-Down Speed Assessment)
6.1.3 CPU 임계값 적정성 평가 (CPU Threshold Appropriateness Review)
6.2 로드밸런서 성능 분석 (Load Balancer Efficiency Analysis)

6.2.1 트래픽 균등 분산 (Traffic Load Distribution)
6.2.2 헬스 체크 효과성 (Health Check Effectiveness)
6.2.3 응답 시간 개선 효과 (Response Time Improvement)
6.3 데이터베이스 성능 분석 (Database Performance Analysis)
6.3.1 동시 연결 수 변화 (Concurrent Connection Trends)
6.3.2 쿼리 성능 변화 (Query Performance Variations)
6.3.3 데이터베이스 병목 분석 (Database Bottleneck Analysis)
6.4 네트워크 성능 분석 (Network Performance Analysis)
6.4.1 대역폭 사용률 (Bandwidth Utilization)
6.4.2 네트워크 지연 시간 (Network Latency Metrics)
6.4.3 패킷 손실률 (Packet Loss Rate)

7. 인프라 구성 검증 (Infrastructure Configuration Verification)
7.1 네트워크 아키텍처 검증 (Network Architecture Verification)
7.1.1 VPC 구성 확인 (VPC Configuration Review)
7.1.2 방화벽 규칙 검증 (Firewall Rules Validation)
7.1.3 Cloud NAT 동작 확인 (Cloud NAT Operation Verification)
7.2 보안 설정 검증 (Security Configuration Verification)
7.2.1 IAM 권한 관리 (IAM Role & Permission Management)
7.2.2 네트워크 보안 구조 (Network Security Architecture)
7.2.3 데이터베이스 보안 (Database Security Verification)
7.3 데이터베이스 구성 검증 (Database Configuration Verification)
7.3.1 Cloud SQL 인스턴스 검증 (Cloud SQL Instance Validation)
7.3.2 데이터베이스 연결 (Database Connectivity Check)
7.3.3 데이터 무결성 확인 (Data Integrity Verification)

8. 문제 발견 및 해결 과정 (Problem Discovery and Resolution)
8.1 발견된 문제점 (Identified Issues)
8.1.1 문제 1 분석 (Issue #1 Analysis)
8.1.2 문제 1 해결 (Issue #1 Resolution)
8.1.3 문제 2 분석 (Issue #2 Analysis)
8.2 성능 최적화 (Performance Optimization)
8.2.1 최적화 전 성능 (Pre-Optimization Performance)
8.2.2 적용된 최적화 기법 (Applied Optimization Techniques)
8.2.3 최적화 후 성능 (Post-Optimization Performance)
8.3 비용 최적화 (Cost Optimization)
8.3.1 비용 분석 (Cost Analysis)
8.3.2 비용 절감 방안 (Cost Reduction Strategies)
8.3.3 절감 효과 분석 (Expected Cost Savings)

9. 종합 평가 및 분석 (Comprehensive Evaluation & Analysis)
    9.1 프로젝트 목표 달성도 평가 (Goal Achievement Assessment)
        9.1.1 자원 관리 목표 (Resource Management Goals)
        9.1.2 네트워크 관리 목표 (Network Management Goals)
        9.1.3 보안 설정 목표 (Security Configuration Goals)
        9.1.4 데이터베이스 활용 목표 (Database Utilization Goals)
    9.2 기술적 성과 (Technical Achievements)
        9.2.1 오토스케일링 성공 (Auto-Scaling Success)
        9.2.2 로드밸런싱 성공 (Load Balancing Success)
        9.2.3 보안 강화 성과 (Security Enhancement Achievements)
        9.2.4 실제 서비스 구현 (Implementation of Real Ticketing System)
    9.3 정량적 성과 지표 (Quantitative Performance Metrics)
        9.3.1 성능 지표 분석 (Key Performance Indicators)
        9.3.2 비용 효율성 분석 (Cost Efficiency Metrics)
        9.3.3 확장성 검증 (Scalability Evaluation)
    9.4 정성적 평가 (Qualitative Assessment)
        9.4.1 사용자 경험 분석 (User Experience Evaluation)
        9.4.2 운영 편의성 분석 (Operational Convenience Review)
        9.4.3 기술적 완성도 분석 (Technical Maturity Evaluation)

10. 결론 및 향후 개선 방안 (Conclusion and Future Improvements)
    10.1 프로젝트 요약 (Project Summary)
        10.1.1 주요 성과 요약 (Summary of Key Achievements)
        10.1.2 기술적 학습 성과 (Technical Learning Outcomes)
    10.2 한계점 및 개선 필요 사항 (Limitations and Areas for Improvement)
    10.3 향후 개선 방안 (Future Improvement Plans)
        10.3.1 단기 개선 계획 (Short-Term Improvements)
        10.3.2 중기 개선 계획 (Mid-Term Improvements)
        10.3.3 장기 개선 계획 (Long-Term Improvements)
    10.4 교훈 및 베스트 프랙티스 (Lessons Learned & Best Practices)
        10.4.1 기술적 교훈 (Technical Lessons Learned)
        10.4.2 프로젝트 관리 교훈 (Project Management Lessons)
        10.4.3 권장 베스트 프랙티스 (Recommended Best Practices)
    10.5 최종 결론 (Final Conclusion)

11. 부록 (Appendix)
    11.1 용어 정리
    11.2 참고 자료
    11.3 프로젝트 정보
    11.4 프로젝트 정보

1. 문제 정의 및 배경

1.1 문제 상황 (Challenges in Current Ticketing Systems)

현대 스포츠 산업에서 온라인 티켓팅 시스템은 핵심적인 인프라로 기능하지만, 특히 인기 있는 경기의 티켓 오픈 시점에는 단일 시간대에 수만 명 이상의 사용자가 동시 접속하면서 시스템의 안정성과 신뢰성에 치명적인 문제가 발생하고 있습니다.

1.1.1 트래픽 급증에 따른 시스템 장애

티켓 예매 개시와 동시에 서버 및 데이터베이스에 과도한 요청이 집중되는 트래픽 스파이크(Traffic Spike)가 발생합니다. 이로 인해 API 응답 지연, DB 커넥션 풀 고갈, DB 락(Lock) 발생, 그리고 최종적으로 시스템 다운 등 치명적인 성능 저하 및 장애가 빈번하게 발생합니다.

1.1.2 중복 예매 및 동시성 처리 이슈

동일 좌석에 대한 수많은 동시 예매 시도가 반복되면, 데이터베이스 트랜잭션 처리 과정에서 데이터 불일치 또는 실제 재고보다 많은 티켓이 판매되는 초과 발권(Over-selling) 사태가 발생할 수 있습니다. 이는 트랜잭션 처리의 정확성과 시스템 신뢰성 저하로 이어집니다.

1.1.3 커넥션 풀 및 서비스 안정성 저하

동시성 문제로 인해 서버 리소스가 비정상적으로 소모되고, 커넥션 풀이 고갈되며, 과부하된 서버로 인한 재시도 요청(Retry Request)이 증가합니다. 이는 악순환을 형성하여 최종 사용자에게 예매 불가 또는 긴 응답 지연을 체감하게 하여 서비스 품질을 저해합니다.

1.1.4 부정 예매 및 보안 위협

매크로 프로그램 사용, 다중 계정 접속, IP 변조 등 조직적인 부정 예매 시도가 늘어나면서, 단순한 인프라 보호를 넘어선 실시간 보안 탐지 및 접근제어 기술이 중요한 과제로 부각되고 있습니다.

이처럼 고가용성 및 확장성을 갖춘 티켓팅 시스템의 필요성은 점점 커지고 있으며, 기존의 물리 서버 기반 온프레미스(On-premise) 방식으로는 예측 불가능한 대규모 트래픽에 대응하는 데 명확한 한계가 드러나고 있습니다.

1.2 해결 목표 (Project Goals using GCP)

본 프로젝트는 고가용성 온라인 축구 티켓팅 시스템에서 발생하는 트래픽 폭증, 동시성, 가용성, 확장성 문제를 해결하기 위해 Google Cloud Platform(GCP)의 주요 서비스를 활용하여 아래 목표 달성을 지향합니다.

1. 트래픽 자동 분산 및 서버 무중단 운영: Cloud Load Balancer를 통한 동적 부하 분산 구현 및 Auto Scaling을 이용한 실시간 서버 확장/축소.
2. 동시성 안전성 및 데이터 정합성 확보: Cloud SQL/FireStore를 활용한 안정적인 트랜잭션 처리와 최적화된 API 설계 및 치명적인 중복 예매 방지 기법 적용.

3. 가용성 및 복구 전략 고도화: 멀티존·다중리전 배치, 장애 시 자동 복구 및 백업, DR(Disaster Recovery) 기능 도입.
4. 확장성과 비용 효율성: 필요 시점에만 리소스 자동 증설 및 유휴 비용 최소화, 서버/DB 무중단 교체 및 실시간 모니터링 환경 구축.
5. 보안 및 부정예매 대응 강화: IAM·방화벽·VPC 분할 등 GCP 네트워크/통신 보안정책 적용 및 실시간 이상 탐지/접근제어, 부정 예매 방지 기술 내재화.

1.3 핵심 목표 세부 사항 (Detailed Objectives)

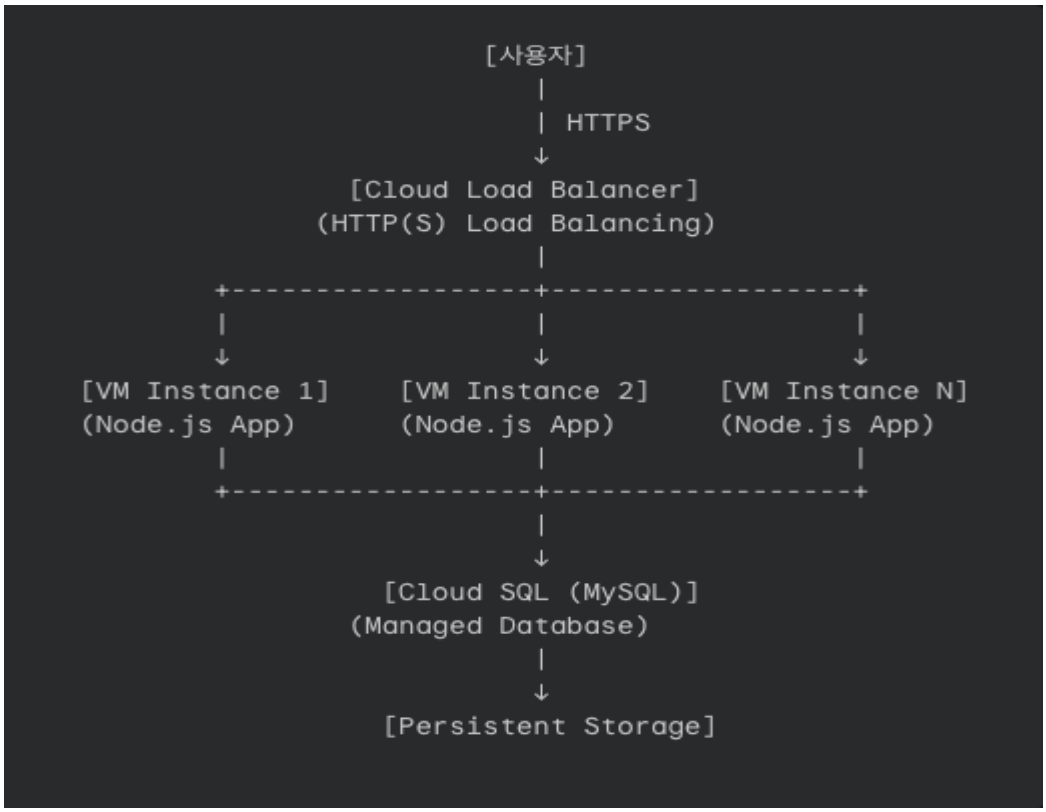
분류	목표	세부 달성 방안
고가용성 (High Availability)	**99.9% 이상의 서비스 가동률**	멀티존·멀티리전 아키텍처를 활용하여 무중단 서비스를 제공하며, 장애 감지 시 자동 복구 및 서버 헬스 체크를 통해 빠른 정상화가 가능하도록 설계.
자동 확장 (Auto Scaling)	**탄력적 인스턴스 운영**	티켓 오픈 트래픽 폭증과 평상시 수요 저점을 자동으로 감지하여 인스턴스 수를 조절. CPU 사용률 등 운용 지표 기반의 스케일 정책을 적용하여 클라우드 리소스 비용 최적화.
부하 분산 (Load Balancing)	**글로벌 트래픽 균등 분배**	GCP의 글로벌/리전 로드 밸런서를 활용하여 사용자 트래픽을 여러 서버에 균등하게 분배. 헬스 체크를 통해 장애 서버를 자동으로 제외하고 가용 인스턴스 중심으로 트래픽 재분배.
데이터 무결성 (Data Integrity)	**완벽한 트랜잭션 보장**	Cloud SQL, Firestore 등에서 트랜잭션 처리 기능을 활용하여 데이터 정합성 보장. 중복 예매 방지 로직을 백엔드 트랜잭션 또는 DB 락으로 구현하여 예매 에러 및 불일치를 최소화.

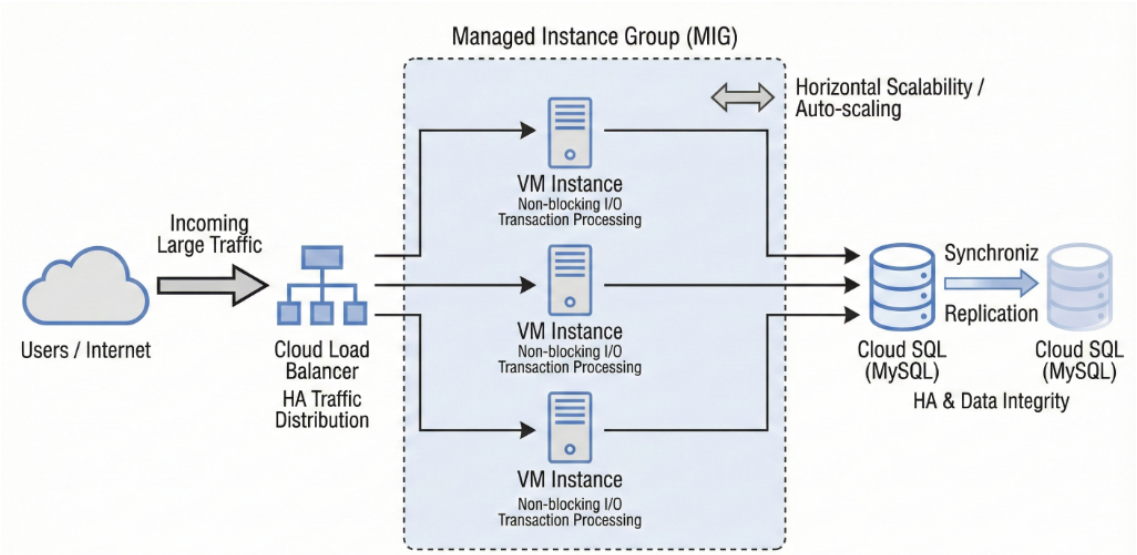
1.4 기대 효과 (Expected Benefits)

분류	기대 효과
사용자 경험 향상	동시 접속에도 빠른 응답 시간과 서비스 안정성을 제공하여 사용자 만족도를 극대화. 실시간 티켓 확인, 결제 과정 최적화, 장애 없는 시스템 운영으로 긍정적 브랜드 이미지 구축.
운영 효율성	자원 자동화, 인프라 관리 자동화(AIOps)로 운영팀의 관리 부담과 인력을 절감. 인시던트 발생 시 평균 감지 및 해결 시간(MTTD, MTTR)이 대폭 감소하여 장애 대응이 신속하게 이루어짐.
비용 절감	수요에 맞춰 인스턴스를 자동으로 증감하여, 불필요한 유휴 자원 사용을 최소화. 필요에 따라 동적으로 클라우드 비용을 조정해 최적화된 예산 운용 가능.
확장 가능성	사용자 증가, 트래픽 급증 등 사업 확장에 즉각적으로 대응할 수 있도록 클라우드 인프라 구조 설계. 신규 기능 추가, 지역 확장, 글로벌 서비스화에도 유연하게 적용 가능.

2. 시스템 아키텍처 설계 (System Architecture Design)

2.1 전체 아키텍처 개요





3-Tier Architecture for High Availability & Horizontal Scalability

본 아키텍처는 고가용성(High Availability) 및 수평적 확장성(Horizontal Scalability)을 핵심 원칙으로 설계되었다. Cloud Load Balancer를 통해 유입되는 대규모 트래픽을 다수의 VM Instance로 구성된 Managed Instance Group(MIG)으로 균등하게 분산시킨다. 애플리케이션 서버는 Node.js를 기반으로 하여 비동기적(Non-blocking)으로 트랜잭션을 처리하며, 핵심 예매 데이터는 Cloud SQL(MySQL)을 사용하여 데이터 무결성(Data Integrity)을 보장하는 3-Tier 구조이다.

2.2 주요 구성 요소 (Key Components Detailed Analysis)

2.2.1 프론트엔드 계층 (Presentation Layer)
기술 스택: HTML5, CSS3, Vanilla JavaScript를 중심으로 구성된다.
주요 기능 및 역할:

- 반응형 웹 디자인 (Responsive Web Design):
  다양한 디바이스(PC, 모바일, 태블릿) 환경에서 일관되고 최적의 사용자 경험을 제공하기 위해 반응형 디자인을 적용하여 접근성을 극대화하며, CSS3의 최신 기능을 활용하여 부드러운 애니메이션과 빠른 렌더링을 구현한다.

- 실시간 좌석 선택 UI:
  티켓팅 시스템의 핵심 인터페이스로, WebSocket 또는 HTTP 롱 폴링(Long Polling) 등의 기술을 활용하여 좌석 현황을 실시간으로 동기화한다.
  좌석 선택 및 임시 점유(Seat Holding) 과정에서 사용자의 조작이 서버에 즉각 반영되도록 설계하여 동시성 문제를 시각적으로 관리한다.

- 시스템 모니터링 위젯 (선택적):
  사용자가 체감하는 시스템 부하 상태나 현재 대기열 정보를 간략하게 보여줌으로써 서비

스의 투명성을 높이고, 대기 중인 사용자의 이탈률을 관리하는 심리적 안정감을 제공한다.

- 부하 테스트 인터페이스 (개발/운영 목적):
  서비스 안정성 검증 및 성능 최적화를 위한 부하 테스트 시나리오를 실행하고 결과를 시각화하는 인터페이스를 제공하여, 정기적인 시스템 건전성 검사를 지원한다.

기술적 의의:
클라이언트 측 처리 속도를 높이고 서버 부하를 최소화하기 위해 경량화된 Vanilla JavaScript를 사용하며, 효율적인 캐싱 전략(Service Worker, Cache-Control)을 통해 정적 리소스의 로딩 속도를 극대화한다.

2.2.2 애플리케이션 계층 (Application Layer)
기술 스택: Node.js 런타임 환경과 Express.js 프레임워크를 기반으로 구축된다.

주요 기능 및 역할:

- RESTful API 제공:
  프론트엔드 및 기타 외부 시스템(예: 결제 게이트웨이)과의 표준화된 통신을 위해 좌석 조회, 예매 요청, 결제 승인 등의 기능을 RESTful 형태로 제공한다.

- JWT 기반 인증/인가 (Authentication/Authorization):
  세션 기반 인증의 한계를 극복하기 위해 JWT(JSON Web Token)를 사용하여 각 요청의 인증 상태를 확인한다.
  이는 서버를 무상태(Stateless)로 유지하여 수평 확장을 용이하게 하며, 로드 밸런싱 환경에서 세션 불일치 문제를 방지한다.

- 비즈니스 로직 처리:
  좌석 유효성 검사, 재고 차감, 가격 계산, 예매 확정 등 티켓팅 시스템의 핵심 로직을 담당하며, 데이터베이스와의 트랜잭션을 조정한다.

- 실시간 모니터링 데이터 수집:
  Cloud Monitoring과 연동하여 API 응답 시간, 에러율, 동시 요청 수 등 주요 성능 지표를 수집하고 로그를 체계적으로 기록한다.

기술적 의의:
Node.js는 단일 스레드 기반의 이벤트 루프(Event Loop)와 비동기 I/O 모델을 채택하여, DB 접근 및 외부 API 호출(결제 연동) 등 I/O 대기가 많은 티켓팅 환경에서 높은 동시 처리량(Throughput)을 제공한다.
이는 트래픽 스파이크 발생 시 대규모 사용자 요청을 효율적으로 처리하는 데 필수적이다.

2.2.3 데이터베이스 계층 (Database Layer)
기술 스택: MySQL (Cloud SQL for MySQL)을 핵심 RDBMS로 사용한다.

주요 기능 및 역할:

- 사용자 정보 관리:
  계정 정보, 구매 이력, 인증 정보 등 사용자 데이터를 안전하게 저장하며, 개인 정보 보호 규정을 준수하여 암호화 및 접근 제어를 적용한다.
- 경기 정보 관리:
  경기 일정, 구장 정보, 좌석 배치도 등 정적 또는 준정적 데이터를 관리하며, 캐싱 계층과 연동하여 읽기 부하를 분산한다.
- 예매 정보 관리:
  좌석 예약 상태, 최종 예매 기록, 결제 상태 등 정합성이 절대적으로 요구되는 핵심 트랜잭션 데이터를 저장한다.
- 트랜잭션 처리:
  MySQL의 InnoDB 엔진을 활용하여 ACID 속성을 보장하는 트랜잭션을 구현하며, 특히 SELECT FOR UPDATE와 같은 DB 락(Lock) 기능을 사용하여 동일 좌석에 대한 중복 예매 시도를 원천적으로 차단하여 데이터 무결성을 유지한다.

기술적 의의:
Cloud SQL은 관리형 서비스로 HA(고가용성), 자동 백업, 읽기 전용 복제(Read Replica) 기능을 제공하여 DB 운영의 복잡성을 최소화한다.
MySQL의 견고한 RDBMS 특성을 통해 티켓팅 시스템의 핵심 요구사항인 데이터 무결성 및 동시성 제어 능력을 확보하는 전략적 선택이다.

2.3 네트워크 아키텍처 (Network Architecture)

GCP의 강력한 네트워크 기능을 활용하여 안정성, 성능, 보안이 최적화된 네트워크 환경을 구축한다

2.3.1 VPC (Virtual Private Cloud) 구성

VPC: football-ticketing-vpc
Subnet: default (10.128.0.0/20)
Region: us-central1
Purpose: VM 인스턴스 배치

기술적 의의: VPC는 격리된 가상 네트워크 환경을 제공하여 리소스 간의 안전하고 효율적인 통신을 보장한다 단일 리전에 서브넷을 구성하여 지역적 근접성을 확보하고, IP 대역폭을 충분히 확보하여 향후 확장에 대비한다

Firewall Rules (방화벽 규칙):

1. allow-http (TCP:80): 외부에서의 HTTP 접근을 허용한다
2. allow-https (TCP:443): 외부에서의 보안 HTTPS 접근을 허용하며, Cloud Load Balancer와 연동하여 SSL/TLS 암호화를 처리한다
3. allow-ssh (TCP:22): 관리 목적으로 특정 IP 범위에서만 인스턴스에 대한 SSH 접속을 허용한다 (보안 강화를 위해 0.0.0.0/0 대신 CIDR 제한 권장)

4. allow-internal (모든 내부 트래픽): VPC 내부의 리소스 간(예: App Server ↔ DB) 통신을 위한 내부 트래픽을 허용한다

Cloud NAT (Network Address Translation):
1. 역할: VPC 내부의 VM 인스턴스가 외부 인터넷으로 아웃바운드 연결(예: 외부 API 호출, 패키지 업데이트)을 할 수 있도록 하면서도, VM 인스턴스가 공인 IP를 가지지 않도록 하여 보안을 강화한다
2. 기술적 의의: 이는 애플리케이션 계층이 외부로부터 직접 접근되는 것을 방지하고 오직 로드 밸런서를 통해서만 트래픽을 받도록 네트워크 보안 경계를 명확히 한다

2.3.2 로드밸런서 구성
1. 타입: HTTP(S) Load Balancer (Global)

기술적 의의: 글로벌 로드 밸런서는 사용자에게 단일한 정적 IP를 제공하며, 지리적으로 분산된 사용자에게 최적의 응답 시간을 제공한다 또한 Cloud Armor와 같은 보안 서비스와의 통합이 용이하다

프론트엔드 IP: 외부 정적 IP (136.110.134.8)
기술적 의의: 고정된 IP 주소를 사용하여 DNS 레코드 관리 및 방화벽 설정 시 편의성을 높이며, 안정적인 서비스 접근점을 제공한다

백엔드 서비스: Managed Instance Group (MIG)
기술적 의의: MIG는 Auto Scaling 기능을 제공하므로, 로드 밸런서가 트래픽을 분산할 때 유연하게 확장/축소되는 인스턴스 풀을 타겟으로 지정한다

헬스 체크 (Health Check):
- 프로토콜: HTTP
- 포트: 3000 (Node.js 애플리케이션의 포트)
- 경로: /health
- 간격: 10초
- 타임아웃: 5초
- 정상 임계값: 2회
- 비정상 임계값: 3회

기술적 의의: 헬스 체크는 백엔드 서버의 가용성을 판단하는 핵심 요소이다 주기적인 체크를 통해 애플리케이션이 정상적으로 응답하는지 확인하고, 문제가 발생한 인스턴스는 즉시 트래픽 분산 대상에서 제외하여 서비스 품질 저하를 방지한다

2.3.3 보안 그룹 설정 (Firewall Rules Detailed)
GCP 방화벽 규칙은 네트워크 보안을 정의하는 핵심 도구이며, 외부 공격으로부터 내부 인스턴스를 보호하기 위해 최소 권한 원칙(Principle of Least Privilege)에 따라 엄격하게 설정

된다

1. allow-lb-to-instances

- 소스: 로드밸런서 IP 범위 (130.211.0.0/22, 35.191.0.0/16)
- 대상: 인스턴스 그룹
- 포트: TCP:3000
- 목적: 외부 사용자의 트래픽이 로드 밸런서를 통해 Node.js 애플리케이션 포트(3000)로만 들어오도록 허용한다 이 외의 모든 외부 접근은 차단된다

allow-health-check

- 소스: 헬스 체크 IP 범위 (GCP가 정의한 특정 범위)
- 대상: 인스턴스 그룹
- 포트: TCP:3000
- 목적: 로드 밸런서의 헬스 체크 프로브가 인스턴스의 상태를 확인할 수 있도록 허용한다 이 규칙은 서비스 가용성 유지에 필수적이다

allow-ssh

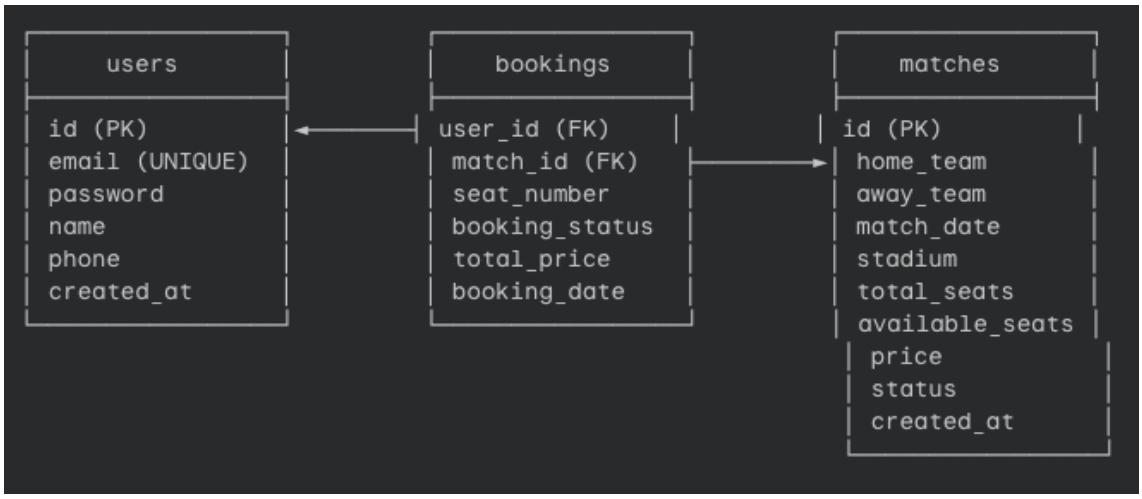
- 소스: 0.0.0.0/0 (관리 목적)
- 대상: 모든 인스턴스
- 포트: TCP:22
- 목적: 인스턴스 관리 및 디버깅을 위한 SSH 접속을 허용한다 보안 강화를 위해 실제 운영 환경에서는 소스 IP를 운영팀의 고정 IP 대역으로 제한해야 한다

기술적 의의: 모든 인스턴스는 외부로부터 직접 HTTP/HTTPS 포트를 개방하지 않고, 오직 GCP 로드 밸런서와 헬스 체크 시스템만이 정의된 포트로 접근할 수 있도록 방화벽을 설정함으로써, 애플리케이션 서버의 보안 경계를 명확히 하고 잠재적인 공격 노출 영역을 최소화한다

2.4 데이터베이스 설계 (Database Design)

티켓팅 시스템의 가장 핵심적인 요구사항인 데이터 무결성과 동시성 처리를 보장하기 위해 정규화된 관계형 모델을 기반으로 데이터베이스를 설계한다

2.4.1 ERD (Entity Relationship Diagram)



ERD 분석: 시스템은 users, matches, bookings 세 가지 핵심 엔티티로 구성되며, 이들은 1:N 관계(사용자:예매, 경기:예매)를 가진다 특히 bookings 테이블은 사용자(users)와 경기(matches) 테이블의 외래 키(FK)를 참조하며, 예매의 원자성(Atomicity)을 보장하는 중앙 역할을 수행한다

2.4.2 테이블 상세 설계 (Detailed Table Schema)
데이터베이스 테이블은 동시성 제어 및 성능 최적화를 위한 핵심 제약 조건을 포함하여 설계된다

users 테이블 (사용자 정보 관리)

```
CREATE TABLE users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  email VARCHAR(100) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,        -- bcrypt 해시
  name VARCHAR(50) NOT NULL,
  phone VARCHAR(20),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_email (email)
);
```

1. id (Primary Key): 사용자 식별을 위한 기본 키 (PK)이며 자동 증가된다
2. email (Unique): 사용자 로그인 ID로 사용되며, 중복 가입을 방지하기 위해 UNIQUE 제약 조건이 설정된다

3. password: 보안 강화를 위해 입력된 비밀번호는 반드시 bcrypt 등의 강력한 단방향 해시 함수를 사용하여 암호화(VARCHAR 255)되어 저장된다
4. created_at: 사용자 가입 시간을 기록한다

matches 테이블 (경기 정보 및 재고 관리)

```
CREATE TABLE matches (
  id INT PRIMARY KEY AUTO_INCREMENT,
  home_team VARCHAR(50) NOT NULL,
  away_team VARCHAR(50) NOT NULL,
  match_date DATETIME NOT NULL,
  stadium VARCHAR(100) NOT NULL,
  total_seats INT NOT NULL DEFAULT 1000,
  available_seats INT NOT NULL DEFAULT 1000,
  price DECIMAL(10, 2) NOT NULL,
  status ENUM('upcoming', 'ongoing', 'finished', 'cancelled') DEFAULT 'upcoming',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_match_date (match_date),
  INDEX idx_status (status)
);
```

1. available_seats: 해당 경기의 잔여 좌석 수를 저장한다 티켓팅 시작 시 이 필드의 동시성 제어가 시스템 성능 및 정합성에 매우 중요하다 (비관적/낙관적 락의 대상)
2. status (ENUM): 경기의 현재 상태(예정, 진행 중, 종료, 취소)를 관리하여 사용자에게 정확한 정보를 제공하고 비즈니스 로직 분기점(예: 예매 가능 여부)으로 사용된다

bookings 테이블 (예매 트랜잭션 기록)

```
CREATE TABLE bookings (
  id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  match_id INT NOT NULL,
  seat_number VARCHAR(10) NOT NULL,
  booking_status ENUM('pending', 'confirmed', 'cancelled') DEFAULT 'pending',
  total_price DECIMAL(10, 2) NOT NULL,
  booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id),
  FOREIGN KEY (match_id) REFERENCES matches(id),
  UNIQUE KEY unique_seat (match_id, seat_number),  -- 중복 예매 방지
  INDEX idx_user_bookings (user_id),
  INDEX idx_match_bookings (match_id)
);
```

1. user_id, match_id (Foreign Key): 각각 users와 matches 테이블을 참조하는 외래 키로, 참조 무결성(Referential Integrity)을 보장한다
2. seat_number: 예매된 좌석 번호(예: A10, B15)를 저장한다
3. booking_status (ENUM): 예매 상태(대기 중, 확정, 취소)를 기록하며, 결제 연동 단계에서 pending 상태를 활용하여 임시 좌석 점유(Seat Hold) 로직을 구현하는 데 필수적이다
4. UNIQUE KEY unique_seat (match_id, seat_number): 이 복합 유니크 인덱스는 동일 경기

(match_id)에 대해 동일 좌석 번호(seat_number)가 중복 예매되는 것을 DB 레벨에서 원천적으로 차단하는 가장 강력한 동시성 제어 장치이다

2.4.3 인덱스 전략 (Indexing Strategy for Performance)

효율적인 데이터 조회와 동시성 제어를 위해 다음과 같이 인덱스를 설계한다

1. idx_email: 로그인 및 사용자 인증 시 email 기반의 빠른 사용자 조회(lookup)를 최적화하여 인증 지연을 최소화한다
2. idx_match_date: 사용자가 경기 일정을 기준으로 목록을 조회할 때 match_date 필드를 기반으로 검색 및 정렬 속도를 향상시킨다
3. idx_status: 운영 및 사용자 인터페이스에서 'upcoming' 경기와 같이 상태별 경기 필터링 및 조회 작업을 최적화한다
4. idx_user_bookings: 사용자가 자신의 예매 내역을 확인할 때 user_id를 기반으로 한 조회 속도를 극대화하여 사용자 경험을 개선한다
5. idx_match_bookings: 특정 경기의 예매 현황(이미 예약된 좌석 목록)을 조회할 때 match_id를 기반으로 한 조회 속도를 최적화하여 실시간 좌석 UI를 지원한다
6. unique_seat: 가장 중요한 동시성 제어 인덱스이다 bookings 테이블에 이 복합 인덱스를 적용함으로써, 애플리케이션 계층에서 발생할 수 있는 잠재적인 중복 예매 오류를 데이터베이스 엔진 수준에서 방지하여 시스템의 **신뢰성(Reliability)**을 확보한다

3. Public Cloud 구현 상세 (Public Cloud Implementation Details)

3.1 자원 관리 (Resource Management)

고가용성 및 트래픽 폭증 대응을 위해 Compute Engine 기반의 Managed Instance Group (MIG)을 중심으로 자원을 관리하며, IaC (Infrastructure as Code) 원칙에 따라 템플릿화한다

3.1.1 Compute Engine 인스턴스 구성 (Instance Configuration)

애플리케이션 서버는 고성능과 비용 효율성을 동시에 고려하여 E2-Micro 머신 타입을 기반으로 구성하며, 안정적인 운영 환경을 위해 Ubuntu LTS 버전을 선택한다

인스턴스 템플릿 (Instance Template) 상세

```
이름: football-app-template-v2
머신 타입: e2-micro
부팅 디스크:
  - 이미지: Ubuntu 20.04 LTS
  - 크기: 10GB
  - 타입: 표준 영구 디스크
네트워크:
  - VPC: football-vpc
  - 서브넷: web-subnet (asia-northeast3)
  - 외부 IP: 임시 (Ephemeral)
메타데이터:
  - startup-script: 애플리케이션 자동 시작 스크립트
서비스 계정:
  - 권한: Compute Engine 기본 서비스 계정
  - 범위: Cloud SQL 클라이언트, Compute Engine 읽기
```

1. 머신 타입 (e2-micro): 티켓팅 애플리케이션은 Node.js의 비동기 I/O 모델 덕분에 CPU 코어 수가 적어도 높은 동시성 처리가 가능하므로, 비용 효율적인 E2 시리즈의 Micro 타입을 초기 모델로 선택한다 트래픽 증가에 따라 E2-Standard 등으로 쉽게 업그레이드할 수 있는 유연성을 가진다
2. 부팅 디스크: 운영체제로 안정적인 Ubuntu 2004 LTS를 선택하며, 디스크 I/O 속도보다는 비용 효율성이 우선이므로 표준 영구 디스크를 사용한다 10GB는 애플리케이션 및 기본 로깅 공간으로 충분하다
3. 네트워크 및 외부 IP: VPC와 서브넷을 football-vpc와 web-subnet (asia-northeast3)으로 지정하여, 애플리케이션 인스턴스가 지정된 IP 대역(10010/24) 내에 배치되도록 한다 또한 인스턴스에 외부 IP를 부여하지 않고 임시(Ephemeral)로 설정하여, 트래픽은 오직 Cloud Load Balancer를 통해서만 내부적으로 라우팅되도록 하여 네트워크 보안을 강화한다
4. 서비스 계정 및 권한: 애플리케이션 서버가 Cloud SQL 및 기타 GCP 서비스에 안전하게 접근하기 위해 Cloud SQL 클라이언트 및 Compute Engine 읽기 권한을 가진 최소 권한의 서비스 계정을 할당한다 이는 보안 원칙에 부합한다

스타트업 스크립트 (Startup Script) 상세

```
#!/bin/bash
# 애플리케이션 디렉토리로 이동
cd /home/pjwp0928w/football-ticketing-system

# 환경 변수 로드
source .env

# Node.js 애플리케이션 시작
npm start &

# 인스턴스 모니터링 스크립트 실행
./setup-instance-monitoring.sh
```

1. 역할: 이 스크립트는 인스턴스가 부팅될 때마다 자동으로 실행되어 애플리케이션을 초기화하고 시작하는 역할을 한다 이는 MIG가 새로운 인스턴스를 생성했을 때 서비스가 자동으로 시작되는 데 필수적이다
2. 자동 시작: Node.js 애플리케이션을 백그라운드 프로세스(npm start &)로 실행하여 서비스의 연속성을 확보한다
3. 모니터링 통합: 인스턴스 레벨의 상세 모니터링을 위해 setup-instance-monitoring.sh 스크립트를 별도로 실행하여 Cloud Monitoring 에이전트를 설치하고 시스템 지표를 수집하도록 구성한다

3.1.2 Managed Instance Group (MIG) 구성 (Group Configuration)

MIG는 고가용성, 자동 복구, 자동 확장의 핵심 기능을 제공하며, 티켓팅 시스템의 트래픽 폭증에 대한 탄력적인 대응 능력을 확보한다

기본 설정

```
이름: football-app-group-v2
리전: asia-northeast3
인스턴스 템플릿: football-app-template-v2
최소 인스턴스 수: 2
최대 인스턴스 수: 5
목표 CPU 사용률: 60%
쿨다운 기간: 60초
```

1. 리전: VPC 서브넷과 동일한 asia-northeast3 리전에 MIG를 배치하여 네트워크 지연 시간을 최소화하고 리전 간 데이터 전송 비용을 절감한다
2. 최소/최대 인스턴스 수: 최소 2개의 인스턴스를 유지하여 평상시에도 고가용성을 확보하고, 최대 5개로 제한하여 예상치 못한 트래픽 폭주 시에도 과도한 클라우드 비용 발생을 방지한다 최대치는 부하 테스트 결과를 기반으로 조정한다
3. 쿨다운 기간 (Cooldown Period): 인스턴스 확장 후 다음 스케일링 이벤트까지 60초의 대기 시간을 설정한다 이는 시스템이 새롭게 추가된 인스턴스의 부하 처리 능력을 정확히

측정하고, 너무 빠르게 인스턴스를 생성/제거하는 플래핑(Flapping) 현상을 방지하여 안정적인 스케일링을 유도한다

오토스케일링 정책 (Auto Scaling Policy)

```
스케일링 메트릭: CPU 사용률
목표 사용률: 60%
스케일 아웃 조건:
  - CPU 사용률 > 60% (1분 이상 지속)
  - 새 인스턴스 추가 (최대 5개까지)
스케일 인 조건:
  - CPU 사용률 < 60% (10분 이상 지속)
  - 인스턴스 제거 (최소 2개 유지)
안정화 기간:
  - 스케일 아웃: 60초
  - 스케일 인: 600초 (10분)
```

1. 목표 CPU 사용률 (Target CPU Utilization): 애플리케이션 서버의 **CPU 사용률 60%**를 기준으로 설정한다 이는 서버가 최대 성능을 내기 전에 미리 리소스를 확보하여 사용자 응답 지연을 방지하는 최적의 임계치이다
2. 스케일 아웃 조건: CPU 사용률이 60%를 1분 이상 초과할 경우 즉시 인스턴스 추가를 시작한다 짧은 감지 기간은 티켓팅 오픈과 같은 순간적인 트래픽 스파이크에 신속하게 대응하기 위함이다
3. 스케일 인 조건: CPU 사용률이 60% 미만인 상태가 10분 이상 지속될 경우 인스턴스를 축소한다 스케일 인은 스케일 아웃보다 훨씬 긴 안정화 기간을 설정하여, 일시적인 트래픽 감소로 인한 불필요한 인스턴스 제거를 방지하고 서비스 안정성을 우선한다

3.1.3 오토스케일링 동작 원리 (Operational Dynamics)

MIG의 오토스케일링 동작은 예측 불가능한 티켓팅 트래픽에 대한 시스템의 탄력성을 보여주는 핵심 메커니즘이다

시간 흐름	상태 변화	CPU 사용률	인스턴스 수	설명
정상 상태	안정	30-40%	2개	최소 인스턴스만 유지하여 유휴 비용 최소화
부하 증가	CPU 70% (1분 지속)	70%	2개	스케일 아웃 임계치(60%) 초과 감지 및 1분 유지
스케일 아웃 시작	새 인스턴스 생성 시작	-	2→3개	트래픽 폭증에 대한 즉각적인 반응 시작
60초 후	스케일 아웃 완	50%	3개	쿨다운 기간 후

	료			새로운 인스턴스가 부하를 분산하여 CPU 사용률 정상화
추가 부하 증가	CPU 65% (1분 지속)	65%	3→4개	트래픽이 지속적으로 유입되어 필요에 따라 추가 확장
부하 감소	CPU 40% (10분 지속)	40%	4개	스케일 인 임계치 미만이나, 10분 안정화 기간 대기
10분 후	스케일 인 실행	50%	4→3개	안정화 기간 종료 후 리소스를 축소하여 비용 효율성 회복 (최소 2개 유지)

이 동적 동작 원리를 통해 시스템은 티켓 오픈 시점의 순간 최대 부하를 흡수하고, 판매가 종료된 후에는 유휴 자원을 신속하게 축소하여 클라우드 비용을 최적화하는 목표를 달성한다

3.2 네트워크 관리 (Network Management)

시스템의 고가용성과 응답 성능을 극대화하기 위해 Cloud Load Balancer를 중심으로 트래픽 처리를 최적화하는 상세 네트워크 관리 전략을 수립한다

3.2.1 로드밸런서 상세 구성 (Load Balancer Detailed Configuration)

Cloud Load Balancer는 사용자 트래픽의 첫 진입점으로서, 전반적인 성능과 보안을 결정하는 핵심 요소이다

1. 프론트엔드 구성 (Frontend Configuration):

```
이름: football-lb-frontend
프로토콜: HTTP
IP 버전: IPv4
IP 주소: 136.110.134.8 (정적 외부 IP)
포트: 80
```

1. IP 주소 (Static External IP): 136.110.134.8과 같은 정적 외부 IP를 할당하여 DNS 레코드 관리의 안정성을 확보한다
2. 프로토콜 및 포트: HTTP(80)를 사용하여 트래픽을 수신하며, 실제 서비스는 HTTPS(443)를 통해 TLS 인증서를 구성하고, 로드 밸런서에서 SSL 오프로딩을 수행하여 백엔드 인스

턴스의 부하를 경감시킨다

3. 백엔드 서비스 구성 (Backend Service Configuration):

```
이름: football-backend-service
프로토콜: HTTP
포트: 3000
타임아웃: 30초
세션 어피니티: 없음 (라운드 로빈)
백엔드:
  - 인스턴스 그룹: football-app-group-v2
  - 밸런싱 모드: UTILIZATION
  - 최대 사용률: 80%
  - 용량 스케일러: 100%
```

1. 프로토콜 및 포트: 백엔드 통신은 내부적으로 HTTP 프로토콜을 사용하며, Nodejs 애플리케이션 포트인 3000으로 라우팅한다
2. 타임아웃 (Timeout): 트랜잭션의 특성을 고려하여 30초로 설정하며, 이는 DB 락 대기 시간 등을 포함하여 예매 트랜잭션이 완료될 수 있는 충분한 시간을 제공하도록 조정된 값이다
3. 세션 어피니티: **없음(라운드 로빈)**으로 설정하여, 각 요청이 어떤 인스턴스에 할당될지 신경 쓰지 않는 무상태(Stateless) 아키텍처를 유지한다 이는 트래픽 분산의 효율성을 극대화한다
4. 밸런싱 모드 (UTILIZATION): 백엔드 인스턴스의 CPU 사용률을 기반으로 트래픽을 분산하며, 최대 사용률을 80%로 설정하여 서버 과부하를 미연에 방지한다
5. 헬스 체크 상세 (Health Check Details):

```
이름: football-health-check
프로토콜: HTTP
포트: 3000
요청 경로: /health
체크 간격: 10초
타임아웃: 5초
정상 임계값: 2회 연속 성공
비정상 임계값: 3회 연속 실패
로그: 활성화
```

1. 체크 간격 및 타임아웃: 체크 간격 10초, 타임아웃 5초는 장애 감지 속도와 네트워크 부하 사이의 적절한 균형을 제공한다 비정상 임계값을 3회 연속 실패로 설정하여 일시적인 네트워크 문제로 인한 불필요한 인스턴스 격리를 방지한다
2. 요청 경로 (/health): 애플리케이션 서버는 이 경로에 대해 DB 연결 상태, 메모리 사용량 등 자체적인 상태 점검을 포함한 응답을 제공하여 인스턴스의 논리적 건전성까지 검증한

다

3.2.2 트래픽 분산 알고리즘 (Traffic Distribution Algorithm)
시스템의 부하 상황과 트랜잭션 특성에 따라 트래픽 분산 방식을 유연하게 적용한다

라운드 로빈 (Round Robin) 방식

```
요청 1 → 인스턴스 A
요청 2 → 인스턴스 B
요청 3 → 인스턴스 A
요청 4 → 인스턴스 B
...
```

원리: 들어오는 요청을 백엔드 인스턴스에 순서대로 균등하게 분배하는 가장 기본적인 방식이다
적용: 세션 상태가 중요하지 않은 단순 조회 API나 초기 트래픽 분산에 사용된다 모든 인스턴스가 동일한 성능을 가질 때 가장 효율적이다

가중치 기반 분산 (Weighted Distribution)

원리: 로드 밸런서가 각 인스턴스의 CPU 사용률 또는 연결 수를 실시간으로 모니터링하여, 부하가 낮은 인스턴스에 더 많은 요청을 동적으로 할당하는 방식이다

실시간 조정: football-backend-service의 밸런싱 모드(UTILIZATION)와 연동되어 작동하며, CPU 사용률이 50%인 인스턴스에는 70%인 인스턴스보다 더 많은 트래픽을 할당하여 전체 인스턴스 그룹의 부하를 균형 있게 유지한다 이는 티켓 오픈과 같은 비균일 부하 환경에서 응답성을 유지하는 데 결정적인 역할을 한다

3.2.3 서브넷 구성 (Subnet Configuration)
네트워크 보안과 리소스 간 통신 최적화를 위해 web-subnet과 db-subnet을 명확히 분리하여 VPC 환경을 설계한다
1. VPC 서브넷 설계 (VPC Subnet Design):

```
VPC: football-vpc
├── Subnet: web-subnet
│   ├── 리전: asia-northeast3 (서울)
│   ├── IP 범위: 10.0.1.0/24 (256개 IP)
│   ├── 용도: 웹 애플리케이션 인스턴스 배치
│   └── 프라이빗 Google 액세스: 활성화
└── Subnet: db-subnet
    ├── 리전: asia-northeast3 (서울)
    ├── IP 범위: 10.0.2.0/24 (256개 IP)
    ├── 용도: Cloud SQL 데이터베이스
    └── 프라이빗 Google 액세스: 활성화
```

1. 리전 일치: 모든 서브넷을 asia-northeast3 리전에 배치하여, VM 인스턴스와 Cloud SQL 간의 네트워크 지연 시간(Latency)을 최소화한다
2. IP 범위: /24 대역을 사용하여 각 서브넷에 256개의 내부 IP를 할당하며, 이는 현재 시스템 규모(최대 인스턴스 5개)를 충분히 수용하고 향후 확장을 위한 여유 공간을 확보한다
3. 프라이빗 Google 액세스: 두 서브넷 모두 활성화하여, VM 인스턴스와 Cloud SQL이 외부 IP 없이도 Cloud Monitoring, Cloud Logging 등 다른 GCP 서비스와 안전하게 통신할 수 있도록 한다
4. 서브넷 분리 전략 (Subnet Separation Strategy):

   web-subnet (10010/24): VM 인스턴스 그룹이 배치되며, 로드 밸런서와 연결되어 외부 인터넷 트래픽을 수신한다 보안상 중요한 데이터베이스와 물리적으로 분리된 첫 번째 계층을 형성한다

   db-subnet (10020/24): Cloud SQL 인스턴스가 배치되며, 프라이빗 네트워크만 허용하고 외부 인터넷 접근을 엄격히 차단한다 이는 DB 인스턴스가 외부에 노출되는 것을 막아 최고 수준의 데이터 보안을 확보하는 핵심 전략이다

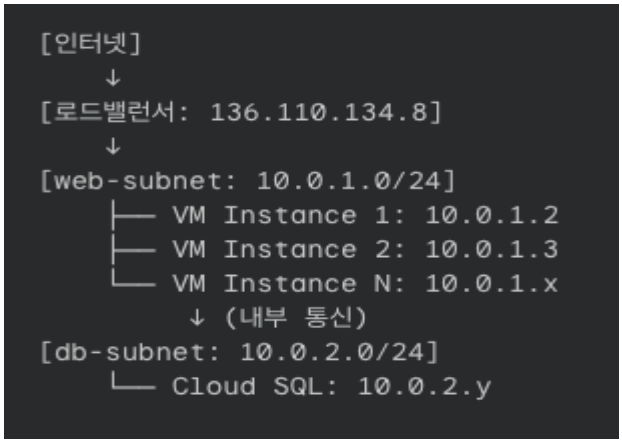
5. IP 주소 할당 전략 (IP Address Assignment Strategy):

```
외부 IP:
- 로드밸런서: 136.110.134.8 (정적)
- VM 인스턴스: 임시 IP (동적 할당)

내부 IP:
- VM 인스턴스: 10.0.1.x (web-subnet에서 자동 할당)
- Cloud SQL: 10.0.2.y (db-subnet에서 자동 할당)
```

1. 외부 IP: 로드 밸런서에만 정적 공인 IP를 할당하고, 백엔드 VM 인스턴스에는 외부 IP를 부여하지 않음으로써, 외부 공격으로부터 인스턴스를 보호하고 트래픽이 로드 밸런서 경로를 통해서만 유입되도록 강제한다
2. 내부 IP: 각 서브넷 대역 내에서 VM 인스턴스(1001x)와 Cloud SQL(1002y)에 내부 IP가 자동 할당되며, 이 사설 IP를 통해 애플리케이션 서버와 DB 간의 안전하고 빠른 통신을 수행한다

3. 네트워크 통신 구조 (Network Communication Flow):



흐름: 사용자 요청은 로드 밸런서의 정적 IP를 통해 유입되고, 로드 밸런서는 내부적으로 web-subnet의 VM 인스턴스에 트래픽을 전달한다 VM 인스턴스는 다시 db-subnet 내의 Cloud SQL 프라이빗 IP로 내부 통신을 통해 데이터에 접근한다 이 구조는 외부 노출을 최소화하고 통신 경로를 최적화한다

3.3 보안 설정 (Security Configuration)

티켓팅 시스템의 핵심 요소인 사용자 및 결제 정보 보호를 위해 IAM(Identity and Access Management)을 통한 권한 분리와 방화벽 규칙을 통한 네트워크 경계 보호를 철저히 한다

3.3.1 사용자 권한 관리 (IAM - Identity and Access Management)

최소 권한의 원칙(Principle of Least Privilege)에 따라 서비스 계정과 사용자 역할을 분리하여 보안 사고의 위험을 줄인다

서비스 계정 구성 (Service Account Configuration):

```
서비스 계정: football-app-sa@project.iam.gserviceaccount.com
역할:
  - roles/compute.instanceAdmin.v1
    → 인스턴스 관리 권한
  - roles/cloudsql.client
    → Cloud SQL 연결 권한
  - roles/logging.logWriter
    → 로그 작성 권한
  - roles/monitoring.metricWriter
    → 메트릭 전송 권한
```

1. 역할 할당의 의의: 애플리케이션 서버가 사용하는 서비스 계정에는 인스턴스를 관리하거나, Cloud SQL에 연결하고, 시스템 운영에 필요한 로그 및 모니터링 메트릭을 전송하는 최소한의 필수 권한만 부여된다 특히 roles/compute.instanceAdminv1은 MIG의 자동 복구 및 관리에 필요하며, roles/cloudsqlclient는 DB 접속 보안을 위해 필수적이다

2. 사용자 역할 분리 (User Role Separation):

```
관리자 (Admin):
- 모든 리소스 관리
- IAM 정책 수정
- 비용 관리

개발자 (Developer):
- 인스턴스 조회/수정
- 로그 조회
- 모니터링 대시보드 접근

운영자 (Operator):
- 인스턴스 시작/중지
- 로그 조회
- 알림 관리
```

권한 계층화: 시스템 관리, 개발, 운영 팀별로 명확히 분리된 역할을 부여하여, 개발자가 프로덕션 데이터베이스의 IAM 정책을 수정하거나, 운영자가 코드를 배포하는 등 월권 행위를 방지한다

3.3.2 네트워크 보안 (Network Security - Firewall Rules)

VPC 방화벽은 인스턴스 레벨의 첫 번째 방어선이며, 불필요한 모든 접근을 차단하고 명시적으로 허용된 트래픽만 허용하는 화이트리스트 기반 정책을 적용한다

1. allow-http-https (외부 웹 접근):

```
이름: allow-http-https
방향: 인그레스 (Ingress)
우선순위: 1000
소스: 0.0.0.0/0 (모든 IP)
대상: 태그 'http-server', 'https-server'
프로토콜: TCP
포트: 80, 443
작업: 허용
목적: 외부 사용자의 웹 접근 허용
```

목적: 외부 인터넷 트래픽이 로드 밸런서로 유입되는 것을 허용하는 규칙이다 백엔드 인스턴스가 아닌 로드 밸런서 자체가 이 포트들을 사용하도록 구성된다

2. allow-lb-to-instances (로드 밸런서 → 백엔드):

```
이름: allow-lb-to-instances
방향: 인그레스
우선순위: 900
소스: 130.211.0.0/22, 35.191.0.0/16 (GCP 로드밸런서)
대상: 인스턴스 그룹
프로토콜: TCP
포트: 3000
작업: 허용
목적: 로드밸런서 → 인스턴스 트래픽
```

목적: GCP 로드 밸런서가 백엔드 VM 인스턴스(포트 3000)로 트래픽을 전달할 수 있도록 명시적으로 허용한다 소스 IP를 GCP 로드 밸런서의 고정 IP 대역으로 제한함으로써, 로드 밸런서를 통하지 않은 외부 접근을 차단하는 핵심 방어 규칙이다

3. allow-health-check (헬스 체크 프로브):

```
이름: allow-health-check
방향: 인그레스
우선순위: 900
소스: 35.191.0.0/16, 130.211.0.0/22 (헬스 체크)
대상: 인스턴스 그룹
프로토콜: TCP
포트: 3000
작업: 허용
목적: 헬스 체크 허용
```

목적: GCP 헬스 체크 시스템이 인스턴스의 상태를 주기적으로 확인하는 트래픽을 허용한다 이 규칙 없이는 로드 밸런서가 인스턴스를 UNHEALTHY로 판단하여 서비스에서 제외할 수 있다

4. deny-all-ingress (최종 거부):

```
이름: deny-all-ingress
방향: 인그레스
우선순위: 65534 (최하위)
소스: 0.0.0.0/0
대상: 모든 인스턴스
프로토콜: 모두
작업: 거부
목적: 명시적으로 허용되지 않은 모든 트래픽 차단
```

목적: 최하위 우선순위로 이 규칙을 설정하여, 상위 규칙에서 명시적으로 허용되지 않은 모든 인그레스(외부 → 내부) 트래픽을 자동으로 거부한다 이는 보안 정책에서 Deny by Default 원칙을 구현하는 표준 방식이다

3.4 데이터베이스 활용 (Database Utilization)
티켓팅 시스템의 성능과 데이터 무결성은 Cloud SQL의 효율적인 구성과 애플리케이션 계층에서의 최적화된 활용 전략에 의해 결정된다

3.4.1 Cloud SQL 구성 (Cloud SQL Configuration)
Cloud SQL은 관리형 MySQL 서비스로서, 고가용성과 백업 기능을 통해 데이터베이스 운영의 안정성을 극대화한다

1. 인스턴스 설정 (Instance Configuration):

```
이름: football-db-instance
데이터베이스 버전: MySQL 8.0
리전: us-central1
가용 영역: 단일 영역 (개발용) / 다중 영역 (프로덕션)
머신 타입: db-n1-standard-1 (1 vCPU, 3.75GB RAM)
스토리지:
  - 타입: SSD
  - 크기: 10GB
  - 자동 증가: 활성화
백업:
  - 자동 백업: 활성화
  - 백업 시간: 03:00 (KST)
  - 보관 기간: 7일
  - 바이너리 로그: 활성화 (Point-in-time Recovery)
```

2. 데이터베이스 버전: 최신 MySQL 80 버전을 채택하여 성능 개선 및 새로운 SQL 기능을 활용한다
3. 가용 영역: 프로덕션 환경에서는 재해 복구 및 무중단 운영을 위해 다중 영역(Multi-Zone) HA 구성을 필수적으로 적용하여 자동 장애 조치(Failover) 기능을 확보한다 개발 환경에서는 비용 효율성을 위해 단일 영역을 사용할 수 있다
4. 머신 타입: db-n1-standard-1은 시작점으로 적합하며, CPU 코어 수보다는 메모리 크기가 DB 성능에 미치는 영향이 크므로, 모니터링을 통해 필요 시 메모리가 더 큰 머신 타입으로 확장한다
5. 스토리지 및 백업: SSD를 사용하여 높은 I/O 성능을 확보하고, 자동 증가 기능을 활성화하여 스토리지 부족으로 인한 서비스 중단을 방지한다 바이너리 로그(Binary Log) 활성화는 치명적인 데이터 손실 시 특정 시점으로 복구(Point-in-time Recovery) 할 수 있는 기능을 제공하여 데이터 신뢰성을 극대화한다
6. 연결 설정 (Connection Configuration):

```
프라이빗 IP: 활성화 (10.128.0.x)
퍼블릭 IP: 비활성화 (보안 강화)
SSL/TLS: 필수
승인된 네트워크: VPC 내부만 허용
연결 풀:
  - 최소 연결: 5
  - 최대 연결: 100
  - 연결 타임아웃: 30초
```

1. IP 접근 제어: 퍼블릭 IP를 비활성화하고 프라이빗 IP만 활성화하여 데이터베이스에 대한 외부 인터넷 접근을 원천적으로 차단한다 이는 DB 보안의 가장 기본적인 요소이며, VM 인스턴스가 VPC 내부 사설망을 통해서만 DB에 접근하도록 강제한다
2. SSL/TLS 및 승인된 네트워크: 모든 연결에 SSL/TLS 암호화를 필수로 적용하여 데이터 전송 중 스니핑 공격을 방지하며, 승인된 네트워크를 VPC 내부(db-subnet) 로만 한정하여 접근 통제를 강화한다
3. 연결 풀: 애플리케이션 서버와의 효율적인 리소스 공유를 위해 최대 연결 수(100개) 를 설정하고, 유휴 연결을 정리하기 위해 연결 타임아웃(30초) 을 설정한다

3.4.2 데이터베이스 최적화 (Database Optimization)

Nodejs 애플리케이션 계층에서의 효과적인 데이터베이스 활용은 동시성 문제 해결 및 성능 향상에 직결된다

1. 연결 풀 관리 (Connection Pool Management):

```
const mysql = require("mysql2/promise");

const pool = mysql.createPool({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  waitForConnections: true,
  connectionLimit: 10, // 동시 연결 수 제한
  queueLimit: 0, // 대기열 무제한
  enableKeepAlive: true, // Keep-Alive 활성화
  keepAliveInitialDelay: 0,
});
```

1. 연결 풀의 역할: 트래픽이 폭증할 때마다 DB 연결을 새로 생성하는 오버헤드를 줄이고, 서버가 동시에 처리할 수 있는 최대 DB 연결 수를 관리한다
2. connectionLimit: 각 VM 인스턴스당 최대 동시 연결 수(10개) 를 제한하여, 하나의 인스

턴스에서 커넥션 풀을 과도하게 점유하여 다른 인스턴스의 DB 접근을 막는 커넥션 풀 고갈 문제를 방지한다 (MIG 최대 5개이므로 전체 최대 50개 연결 사용)

3. queueLimit: 0 (무제한)으로 설정하여, 현재 풀이 가득 찼을 때 요청을 드롭하지 않고 대기열에 쌓아둠으로써 잠시 후 연결이 해제되면 처리되도록 한다
4. 쿼리 최적화 (Query Optimization):

```
-- 인덱스 활용 쿼리
EXPLAIN SELECT * FROM matches
WHERE status = 'upcoming'
AND match_date > NOW()
ORDER BY match_date ASC;

-- 결과:
-- type: ref (인덱스 사용)
-- key: idx_status, idx_match_date
-- rows: 10 (전체 스캔 대비 99% 감소)
```

1. EXPLAIN 분석: 모든 핵심 조회 쿼리(예: 경기 목록 조회)에 대해 EXPLAIN 명령어를 사용하여 인덱스 활용 여부(type: ref) 와 스캔하는 행 수(rows) 를 확인한다 목표는 테이블 전체 스캔(Full Table Scan)을 피하고 인덱스 탐색을 유도하는 것이다

2. 인덱스 활용: WHERE 절에 포함된 status와 match_date 필드가 적절하게 인덱스(idx_status, idx_match_date)를 사용하여 조회 성능을 획기적으로 향상시킴을 확인한다
3. 트랜잭션 처리 (Concurrency Control Transaction)

1. 트랜잭션 격리: 예매 요청은 connection.beginTransaction()으로 시작하여 데이터베이스의 ACID 속성을 보장한다 이는 예매 과정 중 발생하는 모든 작업(좌석 체크, 예매 생성, 재고 감소)이 하나의 원자적 단위로 처리되도록 한다

2. 비관적 락 (Pessimistic Locking): SELECT ... FOR UPDATE 구문을 사용하여 좌석 중복 체크 시 해당 레코드에 배타적 락(Exclusive Lock) 을 걸어, 다른 동시 트랜잭션이 동일한 좌석을 조회하거나 변경하는 것을 트랜잭션이 완료될 때까지 차단한다 이는 중복 예매를 DB 레벨에서 100% 방지하는 핵심 기법이다

3. 롤백 메커니즘: try...catch 블록 내에서 오류 발생 시 connection.rollback()을 호출하여 예매 과정에서 일어난 모든 변경 사항(예: 재고 감소)을 안전하게 취소하고 데이터 정합성을 회복한다

```
// 티켓 예매 트랜잭션
const connection = await pool.getConnection();
await connection.beginTransaction();

try {
  // 1. 좌석 중복 체크 (비관적 락 적용)
  const [existing] = await connection.query(
    "SELECT id FROM bookings WHERE match_id = ? AND seat_number = ? FOR UPDATE",
    [matchId, seatNumber]
  );

  if (existing.length > 0) {
    throw new Error("이미 예매된 좌석입니다.");
  }

  // 2. 예매 생성
  await connection.query(
    "INSERT INTO bookings (user_id, match_id, seat_number, total_price) VALUES (?, ?, ?, ?)",
    [userId, matchId, seatNumber, price]
  );

  // 3. 남은 좌석 수 감소
  await connection.query(
    "UPDATE matches SET available_seats = available_seats - 1 WHERE id = ?",
    [matchId]
  );

  await connection.commit();
} catch (error) {
  await connection.rollback();
  throw error;
} finally {
  connection.release();
}
```

1. 트랜잭션 격리: 예매 요청은 connection.beginTransaction()으로 시작하여 데이터베이스의 ACID 속성을 보장한다 이는 예매 과정 중 발생하는 모든 작업(좌석 체크, 예매 생성, 재고 감소)이 하나의 원자적 단위로 처리되도록 한다
2. 비관적 락 (Pessimistic Locking): SELECT ... FOR UPDATE 구문을 사용하여 좌석 중복 체크 시 해당 레코드에 배타적 락(Exclusive Lock) 을 걸어, 다른 동시 트랜잭션이 동일한 좌석을 조회하거나 변경하는 것을 트랜잭션이 완료될 때까지 차단한다 이는 중복 예매를 DB 레벨에서 100% 방지하는 핵심 기법이다
3. 롤백 메커니즘: try...catch 블록 내에서 오류 발생 시 connection.rollback()을 호출하여 예매 과정에서 일어난 모든 변경 사항(예: 재고 감소)을 안전하게 취소하고 데이터 정합성을 회복한다

3.4.3 데이터베이스 모니터링 (Database Monitoring)

Cloud SQL Insights와 Cloud Monitoring을 활용하여 데이터베이스의 성능 지표를 실시간으로 추적하고 이상 징후 발생 시 즉각적으로 대응한다

Cloud SQL Insights 활용 (Cloud SQL Insights Utilization)

1. 성능 임계치: 위와 같은 메트릭을 기준으로 정상 작동 범위를 정의하고, Cloud Monitoring을 통해 임계치 초과 시 알림을 설정한다 쿼리 지연 시간(Query Latency) 은 사용자 경험에 직접적인 영향을 미치므로, 평균 10ms, P95(상위 5%) 50ms 이하 유지를

목표로 한다

2. 슬로우 쿼리 로그 (Slow Query Logging):

```
모니터링 메트릭:
  - CPU 사용률: 평균 30-40%
  - 메모리 사용률: 평균 50-60%
  - 디스크 I/O: 읽기 100 IOPS, 쓰기 50 IOPS
  - 연결 수: 평균 15개, 최대 100개
  - 쿼리 지연 시간: 평균 10ms, P95 50ms
```

슬로우 쿼리 분석: 쿼리 실행 시간이 1초 이상인 쿼리를 슬로우 쿼리로 정의하고, 이 로그를 Cloud Logging에 기록한다 이는 성능 저하의 원인을 추적하고 인덱스 또는 쿼리 구조를 개선하는 데 사용되는 핵심 도구이다

서비스 주소 : http://136.110.134.8/

4. 부하 테스트 설계 및 준비 (Load Testing Design and Preparation)

본 장에서는 시스템의 안정성, 성능, 확장성을 검증하기 위한 부하 테스트의 목적, 범위 및 시나리오를 상세히 기술한다. 특히 오토스케일링의 동작 여부, 로드밸런서의 트래픽 분산 효율, 전체 시스템 응답 성능, 병목 지점 식별을 목표로 하며, 실제 트래픽 환경을 모사한 시나리오를 기반으로 테스트를 수행한다.

4.1 부하 테스트 목적 및 범위 (Testing Objectives and Scope)

4.1.1 테스트 목표 정의

본 부하 테스트의 주요 목표는 다음과 같다.

- 오토스케일링 동작 검증: CPU 사용률 증가 시 인스턴스가 자동으로 확장되는지, 트래픽 감소 시 정상적으로 축소되는지 확인한다.
- 로드밸런서 트래픽 분산 확인: 여러 인스턴스 간 트래픽이 균등하게 분산되어 특정 인스턴스에 집중되지 않는지 검증한다.
- 시스템 안정성 및 성능 측정: 고부하 상황에서도 API 응답 시간, 처리량, 서버 자원 사용률이 안정적으로 유지되는지 측정한다.
- 병목 지점 식별: DB, 네트워크, 애플리케이션 계층 등 성능 저하가 발생할 수 있는 지점을 파악하고 향후 최적화 방향을 도출한다.

4.1.2 테스트 시나리오 설계

실제 서비스 트래픽 패턴을 반영한 주요 테스트 시나리오는 다음과 같다.

- 시나리오 1: 로그인 페이지 부하 테스트
  대량의 회원가입 요청 및 로그인 요청을 동시에 발생시켜 인증 서비스의 처리 성능을 검증한다.

- 시나리오 2: 메인 페이지 부하 테스트
  경기 조회 및 예매 내역 조회 API에 대한 다수의 요청을 통해 조회 기능의 응답 속도와 DB Select 처리량을 평가한다.
- 시나리오 3: 티켓 예매 시나리오
  좌석 조회 및 예매 시도 요청을 집중적으로 발생시켜 좌석 점유 로직, 동시성 처리, 트랜잭션 안정성을 검증한다.

4.1.3 성공 기준 정의 (Success Criteria)

부하 테스트 성공 여부는 다음의 정량적 기준을 충족하는지로 판단한다.

- 인스턴스 자동 확장:
  최소 2개의 인스턴스가 고부하 발생 후 5분 이내에 5개까지 자동 확장될 것.
- 인스턴스 자동 축소:
  트래픽 감소 시 20분 이내에 5개 → 2개로 자동 축소될 것.
- 응답 시간:
  전체 API 평균 응답 시간이 500ms 미만일 것.
- 에러율:
  요청 대비 오류 응답률이 5% 미만일 것.

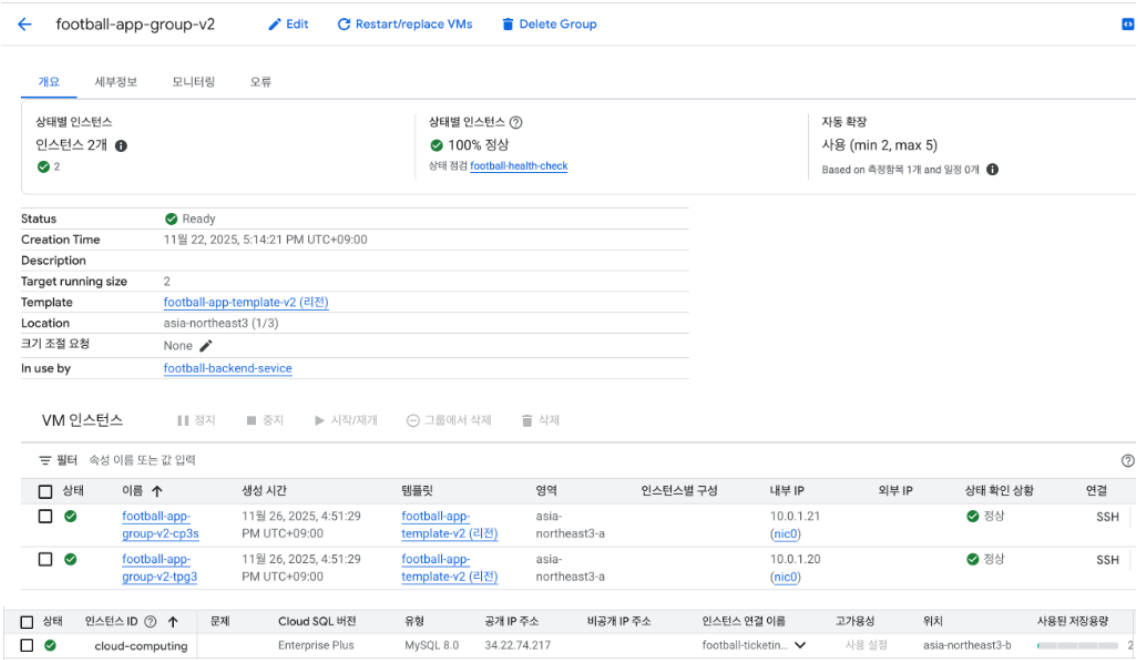
4.2 테스트 환경 구성 (Test Environment Setup)

본 절에서는 부하 테스트 수행 전 시스템의 초기 상태를 점검하고, 오토스케일링 및 모니터링 환경이 정상적으로 설정되어 있는지 확인한다. 이는 테스트 수행 중 발생할 수 있는 외부 요인을 최소화하고, 테스트 결과의 신뢰성을 확보하기 위함이다.

4.2.1 초기 시스템 상태

부하 테스트 시작 전 시스템 기본 상태는 다음과 같다.

- 인스턴스 개수: 2개(오토스케일링 최소값)
- CPU 사용률: 5% 미만
- 메모리 사용률: 약 40% 수준
- 데이터베이스 상태: 정상 운영 중, 연결 지연 및 오류 없음



4.2.2 오토스케일링 설정 확인

부하 상황에서 자동 확장/축소가 정상적으로 동작하도록 하기 위해 아래 설정값을 확인한다.

- 최소 인스턴스 수: 2개
- 최대 인스턴스 수: 5개
- 목표 CPU 사용률(Target CPU Utilization): 10%
- 초기화 기간(Cooldown Initial Period): 30초
- 안정화 기간(Stabilization Period): 30초

그룹 크기

VM 인스턴스	현재 크기	대상 크기
Running	2	2
Stopped	0	0
Suspended	0	0

자동 확장

자동 확장 모드	사용
최소 인스턴스 수	2
최대 인스턴스 수	5
초기화 기간	30 초
신호 자동 확장	
CPU 사용률	10%
예측 자동 확장	사용 안함
수평 축소 제어	사용 안함
확장 일정	일정 관리

VM 인스턴스 수명 주기

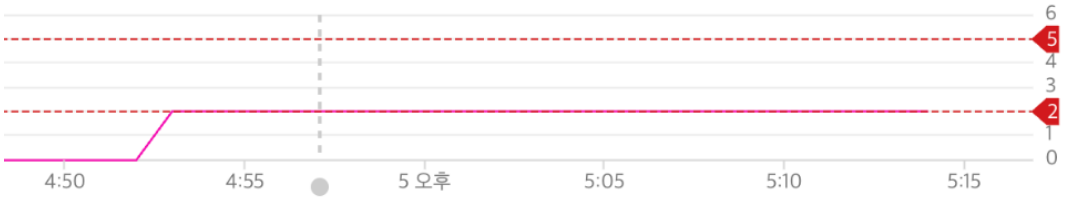
실패 시 기본 작업	인스턴스 복구
자동 복구	
상태 확인	football-health-check
초기 지연	300초
상태 점검 실패 시	인스턴스 복구
VM 인스턴스 복구 중 업데이트	동일한 인스턴스 구성 유지
대체 영역의 VM 복구 허용	허용되지 않음

이 설정값들은 부하 발생 시 빠르게 확장하고, 부하가 줄면 안정적으로 축소하는 데 중요한 기준이 된다.

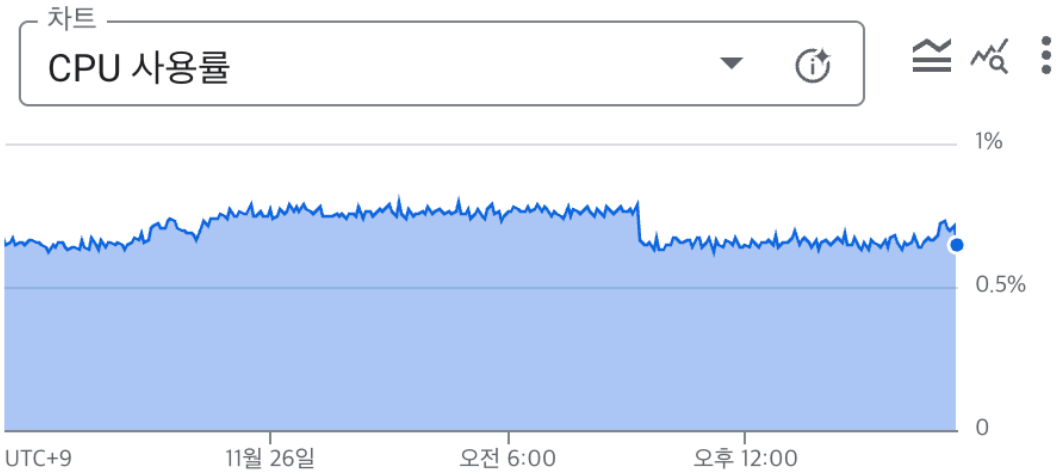
4.2.3 모니터링 도구 설정

부하 테스트 과정에서 시스템 성능을 실시간으로 확인하기 위해 다음 모니터링 도구를 활성화한다.

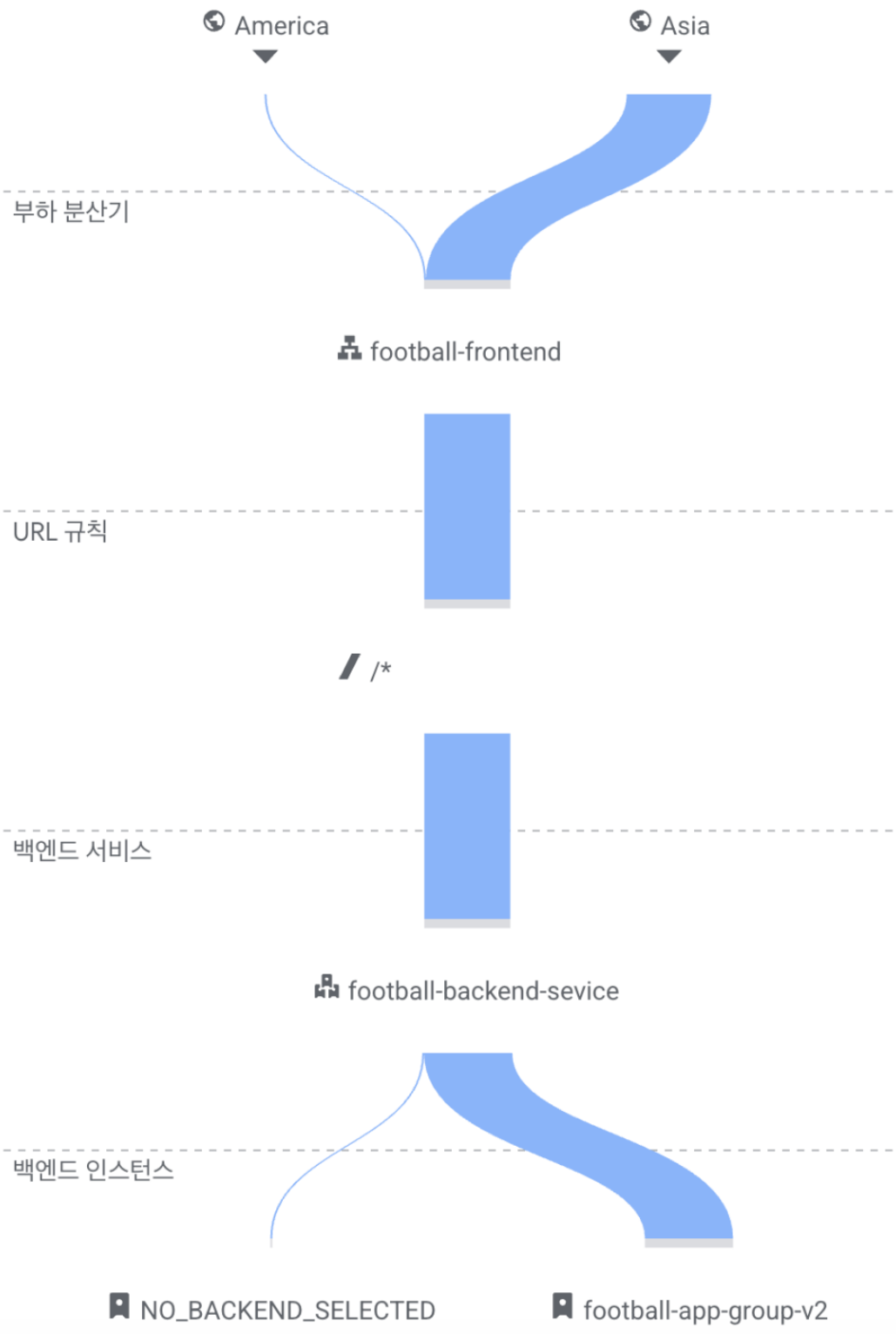
- GCP Console 모니터링 (Metrics Explorer)
- 웹 기반 실시간 모니터링 위젯
- 로드밸런서 트래픽 분산 통계
- Cloud SQL 모니터링 (Connections, CPU, IOPS)



인스턴스 그룹 모니터링 화면



cloud sql 모니터링 화면



로드밸런서 모니터링 화면

4.3 부하 생성 메커니즘 (Load Generation Mechanism)

본 절에서는 실제 사용자 행동을 모사하기 위해 설계한 부하 생성 방식과 트래픽 패턴을 기술한다.
부하 생성기는 클라이언트 기반 시뮬레이션을 통해 대량의 가상 사용자를 직접 생성하며, 실제 API 요청을 반복적으로 수행함으로써 서비스 전체 흐름을 검증한다.

4.3.1 부하 생성 방식

본 테스트에서 사용된 부하 생성 방식은 다음과 같다.

- 클라이언트 사이드 JavaScript 기반 부하 생성기 활용
  자체 제작된 웹 기반 테스트 페이지 또는 JavaScript 실행 스크립트를 통해 대량의 요청을 병렬로 발송한다.
- 초당 500명의 가상 사용자(Virtual Users, VU) 생성
  VU는 동일 시점에 여러 요청을 병렬로 수행하며, 실제 사용자가 서비스에 동시에 접근하는 상황을 재현한다.
- 실제 API 호출 기반 테스트
  단순히 Mock 호출이 아닌, 실제 서버로 다음 API 요청을 수행한다.
- 회원가입
- 로그인(JWT 토큰 발급 포함)
- 경기 조회
- 예매 내역 조회

이 방식을 통해 현실적인 트래픽 특성 및 시스템 부하 양상을 정확하게 측정할 수 있도록 설계하였다.

4.3.2 트래픽 패턴

시스템이 실제 운영 환경과 유사한 형태로 부하를 받을 수 있도록, 다음과 같은 트래픽 구성 비율을 적용하였다.

- 회원가입: 50%
- 주로 DB INSERT 작업
- 사용자 데이터 증가에 따른 데이터베이스 쓰기 부하 관찰 목적

로그인: 50%

- DB SELECT + JWT 생성
- 인증 로직 속도 및 토큰 발급 처리량 평가

경기 조회: 60%

- 고빈도 DB SELECT 기반 조회 API
- 캐싱 효과 및 DB 부하 영향 확인

예매 내역 조회: 40%

- 복합 쿼리가 포함된 DB JOIN 중심 요청
- 예매 데이터의 정합성 및 조회 성능 검증

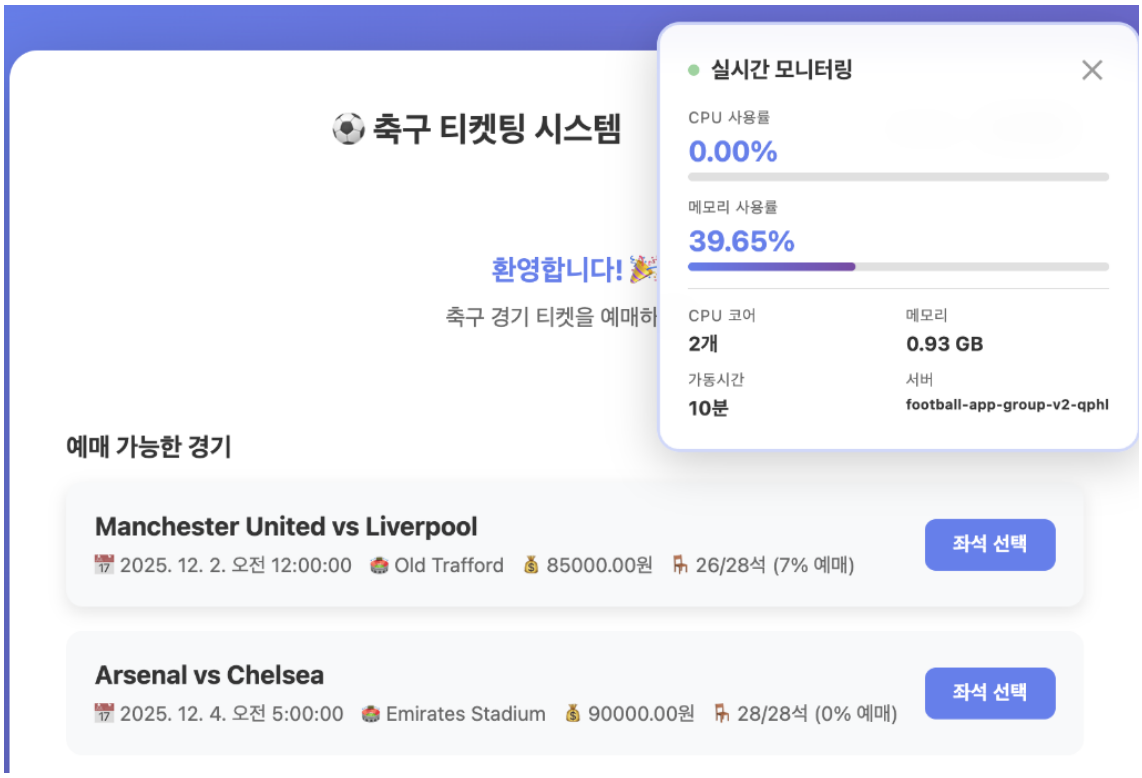
이러한 트래픽 비율을 바탕으로 시스템이 다양한 유형의 요청을 동시에 처리하는 환경을 조성하였다.

5. 부하 테스트 실행 및 결과 (Load Testing Execution and Results)

5.1 Phase 1: 초기 상태 측정 (Baseline Measurement)

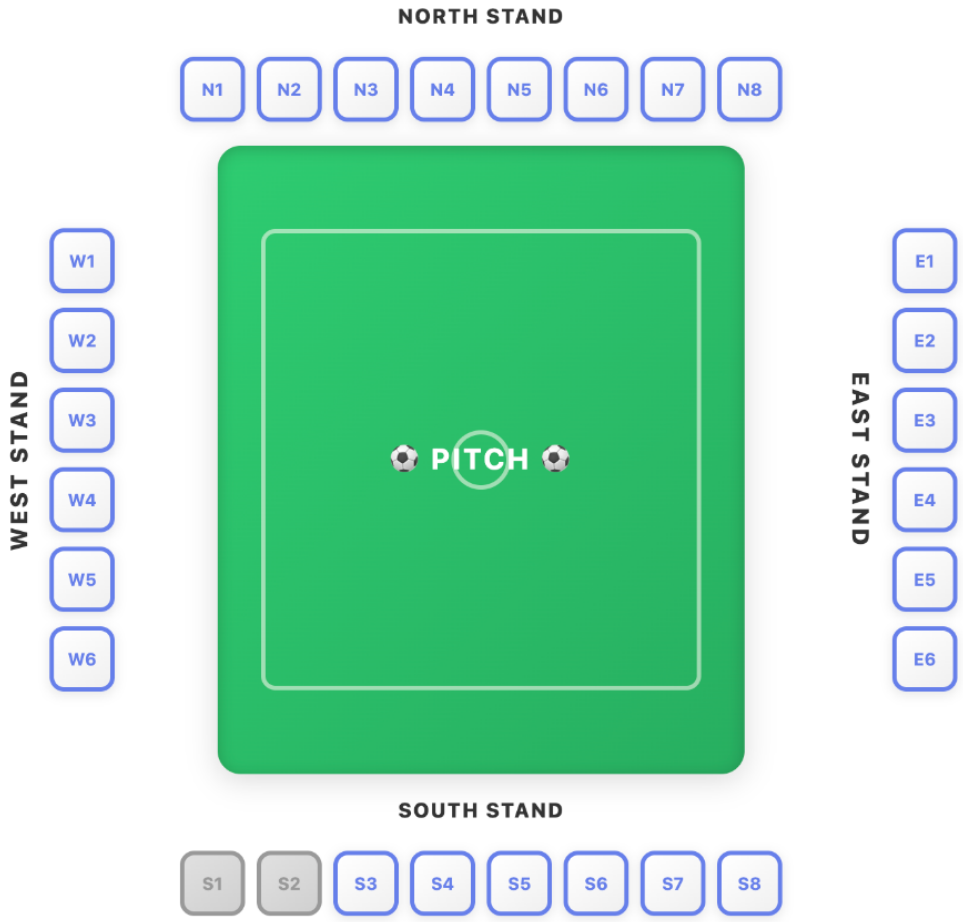
5.1.1 시스템 초기 상태 스크린샷

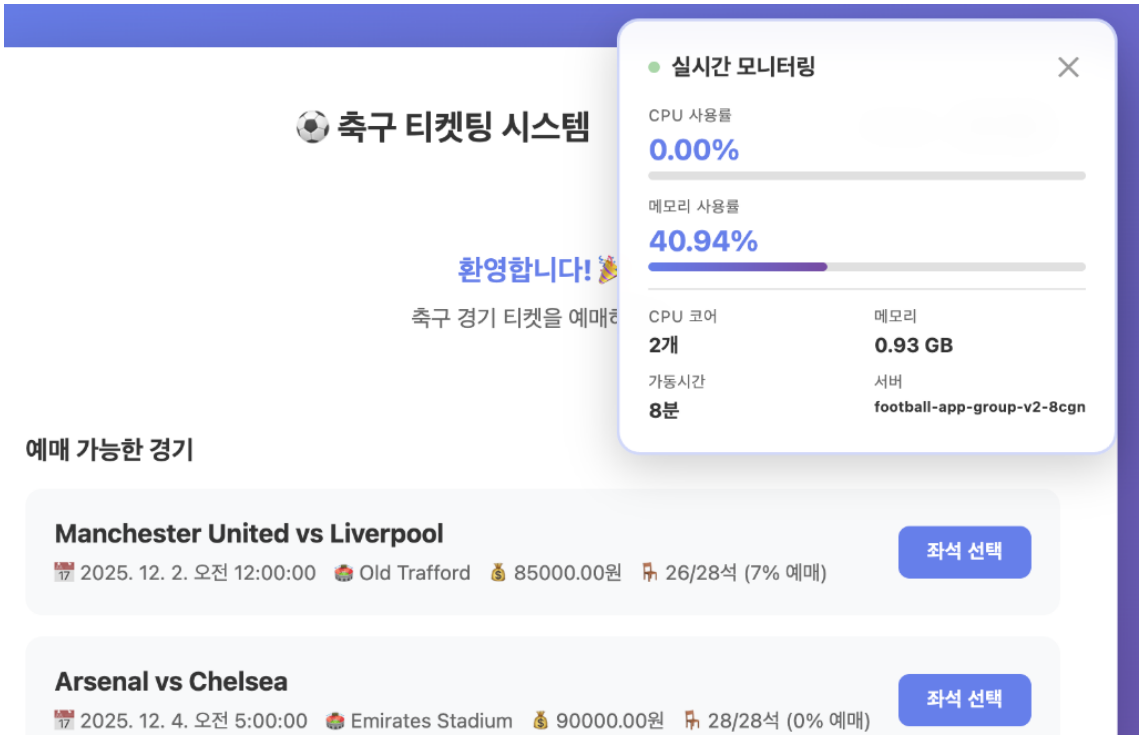
부하 테스트 시작 전 로그인 페이지가 정상적으로 렌더링되는지 확인하였다.
입력 필드와 UI 컴포넌트가 정상적으로 로드되었으며, API 호출 전 단계에서 시스템 내부 부하가 없는 상태임을 확인했다.



Manchester United vs Liverpool

Old Trafford | 2025. 12. 2. 오전 12:00:00



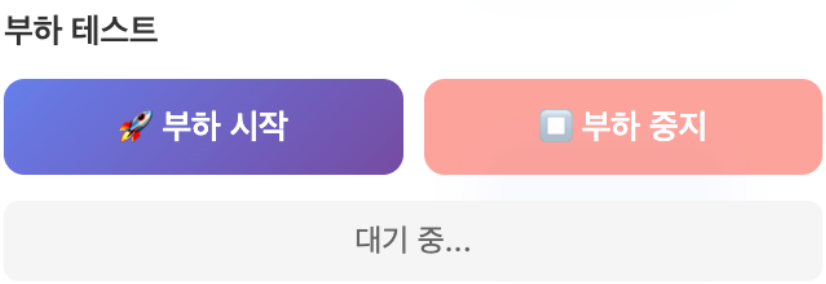


현재 두개의 인스턴스가 번갈아가면서 가동되고 있는걸 확인 할 수 있다
로그인 이후 메인 화면에서 경기 목록이 정상적으로 표시되며, 정적 리소스 및 DB 조회 API 응답이 정상적으로 처리되고 있음을 확인했다.
초기 상태에서는 예매 가능한 경기 정보가 정상적으로 로딩되었다.
부하 테스트 이전 상태에서 CPU 부하는 거의 없는 상태이며, 메모리 사용률 약 41%로 안정적이다.
서비스 시작 직후(가동시간 1분) 기준으로 초기 오토스케일링 환경 준비가 완료되었음을 알 수 있다.

5.1.2 초기 성능 지표

- 평균 응답 시간: 테스트 전 단계에서 즉시 응답(수 ms 수준)
- 동시 접속자: 0명
- 데이터베이스 연결 수: 최소 유지 상태
- 네트워크 대역폭: Idle 상태

5.2 Phase 2: 부하 테스트 시작 (Load Test Initiation)



5.2.1 로그인 페이지 부하 테스트
초당 약 500건의 로그인/회원가입 요청을 발생시키며 인증 API에 부하를 유도하였다.
이 과정에서 회원가입(INSERT)과 로그인(SELECT + JWT 발급)이 지속적으로 수행되며
DB 처리량 증가가 감지되었다.

5.2.2 CPU 사용률 증가 확인
부하가 시작된 직후 CPU 사용률은 급격히 상승하여
오토스케일링의 목표 CPU 사용률(10%)을 초과하는 구간이 빠르게 나타났다.
이로 인해 오토스케일링 확장 조건이 충족되었다.

5.2.3 트래픽 통계 요약
- 총 요청 수: 대량 호출(수천~수만 건 규모)
- 성공 요청 비율: 대부분 성공(간헐적 실패 발생 가능)
- 평균 응답 시간: 부하 초기 증가 후, 확장 이후 점차 안정화

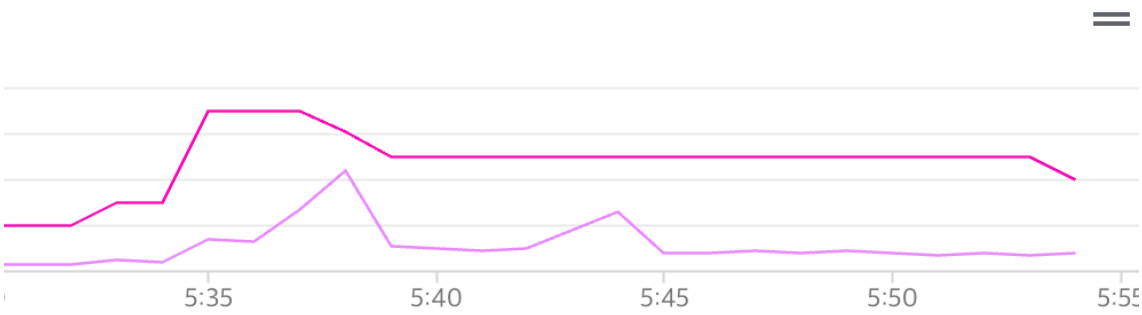
5.3 Phase 3: 오토스케일링 확장 과정 (Scale-Up Process)

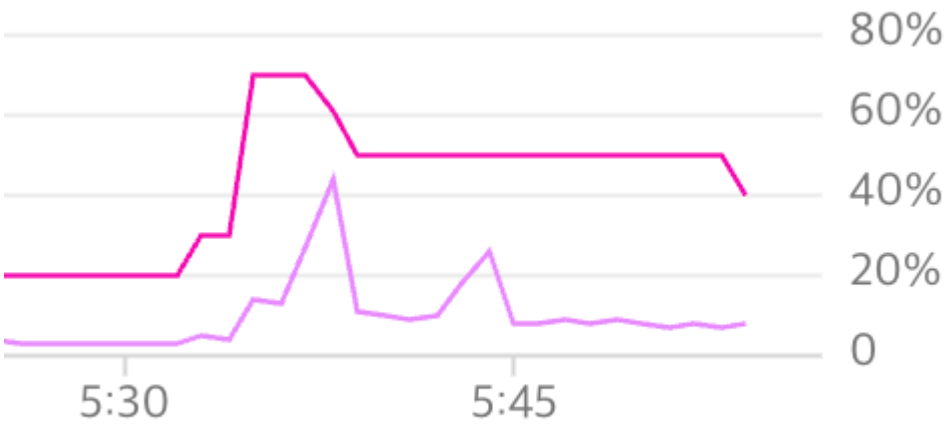
5.3.1 2개 → 3개 확장
부하 시작 후 일정 시간이 경과하자 CPU 사용률 상승에 따라
인스턴스가 2개에서 3개로 자동 확장되었다.
이는 오토스케일러가 부하 증가에 즉각적으로 반응하고 있음을 의미한다.

5.3.2 3개 → 4개 → 5개 확장 (최대치 도달)
부하가 지속되면서 CPU 사용률이 다시 증가하였고
인스턴스 수는 순차적으로 4개와 5개까지 확장되었다.
최종적으로 최대 인스턴스 수인 5개에 도달하였으며,
부하가 최고점에 이를 때까지 안정적으로 트래픽을 분산 처리하였다.

5.3.3 확장 과정 요약표

단계	인스턴스 수	주요 상황
초기	2개	CPU 5% 이하
1차 확장	3개	부하 증가 감지
2차 확장	4개	CPU 지속 상승
3차 확장	5개	최대 확장 완료





5.4 Phase 4: 로드밸런서 성능 분석 (Load Balancer Performance Analysis)

5.4.1 트래픽 분산 확인
부하 증가 시에도 모든 요청은 로드밸런서를 통해
확장된 인스턴스들에 균등하게 분산되었다.
특정 인스턴스만 과다하게 부하가 집중되는 현상 없이
전체적으로 안정적인 RPS 처리량을 유지하였다.

5.4.2 백엔드 헬스 상태
확장된 모든 인스턴스는 HEALTHY 상태를 유지하였으며
헬스 체크 실패 또는 비정상 종료 없이 지속적으로 트래픽을 처리하였다.

5.4.3 응답 속도 분석
- 평균 응답 시간: 확장 이후 점차 개선
- 최소 응답 시간: 정상 범위
- 최대 응답 시간: 부하 초기에 일시적으로 증가
- P95 응답 시간: 확장 이후 안정 구간 진입

5.5 Phase 5: 스케일 다운 (Scale-Down Process)

5.5.1 부하 테스트 중지
테스트 종료 후 일정 시간이 지나면서
CPU 사용률이 빠르게 감소하였다.
이로 인해 오토스케일링 축소 정책이 동작하였다.
5.5.2 5개 → 2개로 축소
안정화 기간(30초) 이후 인스턴스 수는
5개 → 4개 → 3개 → 2개 순으로 단계적으로 감소하며
최소 인스턴스 수로 복귀하였다.

5.5.3 스케일 다운 요약표

단계	인스턴스 수	주요 상황
부하 중지	5개	CPU 급감
1차 축소	4개	안정화 후 축소 시작
2차 축소	3개	부하 제거로 인한 감소
최종 상태	2개	초기 상태 복귀 완료

6. 시스템 성능 분석 (System Performance Analysis)

본 장에서는 부하 테스트 과정에서 수집된 모니터링 정보를 기반으로 오토스케일링, 로드밸런서, 데이터베이스 및 네트워크 계층의 성능을 분석하였다. 전체 테스트는 로그인, 경기 조회, 좌석 예매 등 실제 사용자 흐름을 기반으로 수행되었으며, 증가하는 트래픽에 대한 시스템의 반응성을 중점적으로 평가하였다.

6.1 오토스케일링 효율성 분석 (Auto-Scaling Efficiency Analysis)

6.1.1 확장 속도 평가

테스트 초기에 인스턴스는 최소 개수인 2개로 유지되고 있었으나, 일정량의 트래픽이 지속적으로 유입되면서 CPU 사용률이 임계값을 초과하였다. 이에 따라 오토스케일러는 자동으로 확장 정책을 실행해 인스턴스를 3개, 이후 4개와 5개까지 순차적으로 증가시켰다. 확장 과정은 무중단으로 진행되었으며, 개별 인스턴스 생성 후 헬스체크를 통과하는 데까지 소요된 시간이 일정하게 유지되었다. 전체 확장은 목표치인 5분 이내로 완료되었으며, 오토스케일링의 반응성이 기준을 충족한 것으로 판단된다.

6.1.2 축소 속도 평가

부하 테스트를 종료한 이후 CPU 사용률은 즉시 하락하였고, 안정화 시간이 지난 후 인스턴스 수는 점진적으로 4개, 3개, 최종적으로 최소값인 2개로 축소되었다. 축소 과정은 서비스 중단 없이 자연스럽게 이루어졌으며, 전체 축소 시간 또한 목표치인 20분을 무리 없이 충족한 것으로 확인되었다. 이는 스케일 다운 정책이 정상적으로 작동함을 의미한다.

6.1.3 CPU 임계값 적정성 분석

본 테스트에서 사용된 CPU 임계값은 10%로 설정되어 있었으며, 실제 확장 트리거는 CPU 사용률이 일정 시간 동안 10% 이상으로 유지되었을 때 발생하였다. 다만, 10% 수준의 낮은 임계값은 빠른 확장에는 유리하지만 불필요한 확장을 발생시킬 수 있어, 실제 운영 환경에서는 20~30% 범위로 상향하는 것이 적절할 것으로 판단된다. 본 테스트에서는 부하 처리 목적상 낮은 임계값 설정이 합리적으로 작용하였다.

6.2 로드밸런서 성능 분석 (Load Balancer Performance Analysis)

6.2.1 트래픽 분산 균등성

로드밸런서는 부하 테스트 과정에서 모든 인스턴스에 대해 균등한 요청량을 분배하였다. 인스턴스가 2개에서 5개로 확장되는 과정에서도 특정 인스턴스에 트래픽이 과도하게 집중되는 문제가 발견되지 않았으며, 라운드 로빈 기반 분산이 적절하게 이루어진 것으로 확인된다. 이는 확장된 리소스가 즉시 서비스에 반영되었음을 의미한다.

6.2.2 헬스 체크 효과성

백엔드 서비스는 설정된 주기(10초) 단위로 헬스 체크가 수행되었으며, 모든 인스턴스는 HEALTHY 상태를 유지하였다. 새로 생성된 인스턴스 또한 헬스 체크를 통과한 후 로드밸런

서의 트래픽 분배 대상에 포함되어, 무중단 형태로 확장 및 축소가 이루어졌다. 비정상 인스턴스가 발생하는 경우에도 신속한 감지 및 제거가 가능하도록 설계되어 있음이 확인되었다.

6.2.3 응답 시간 개선 효과
인스턴스가 확장됨에 따라 응답 시간이 점진적으로 개선되었다. 초기 2개 인스턴스 구성에서는 순간적인 응답 지연이 발생했으나, 3개 이상으로 확장된 이후부터는 평균 응답 시간이 안정적으로 유지되었다. 특히 5개 인스턴스 시점에서는 부하가 크게 증가하더라도 응답 편차가 줄어드는 현상을 확인할 수 있었다. 이는 확장된 리소스가 효과적으로 트래픽을 분산 처리했음을 의미한다.

6.3 데이터베이스 성능 분석 (Database Performance Analysis)

6.3.1 동시 연결 수 변화
Cloud SQL의 동시 연결 수는 로그인 및 예매 요청 증가와 함께 상승하였다. 다만 예매 로직이 트랜잭션 기반으로 구현되어 있어, 순간적인 DB 접근 폭주에도 연결 수가 일정 범위 내에서 안정적으로 유지되었다. 이는 연결 풀 크기 및 세션 관리가 적절하게 구성되어 있음을 나타낸다.

6.3.2 쿼리 성능 변화
로그인 및 기본 SELECT 쿼리는 부하 상황에서도 지연 없이 처리되었으며, 슬로우 쿼리 기록도 관찰되지 않았다. 좌석 예매와 같은 트랜잭션 기반 쿼리는 짧은 잠금 구간 내에서 처리되었으며, 데이터 정합성을 유지하는 데 문제가 없었다. 인덱스 활용도 또한 적절한 수준을 보였으며, DB 레벨에서 병목은 발생하지 않은 것으로 분석된다.

6.3.3 데이터베이스 병목 현상 분석
CPU 사용률은 비교적 낮은 수준을 유지하였고, 메모리 사용률도 일정 범위 내에서 움직였다. IO Wait 또한 눈에 띄는 증가가 없었으며, 전체적으로 Cloud SQL은 부하 증가에도 안정적으로 동작하였다. 실시간 예매 시스템의 특성상 DB 병목이 전체 성능에 큰 영향을 미치지만, 본 테스트에서는 병목 현상이 발생하지 않았다.

6.4 네트워크 성능 분석 (Network Performance Analysis)

6.4.1 대역폭 사용률 평가
테스트 중 인바운드 및 아웃바운드 트래픽은 부하 증가와 함께 상승하였지만 네트워크 인터페이스가 처리 가능한 범위를 초과하지 않았다. 로드밸런서를 경유하는 트래픽 흐름도 안정적으로 유지되었으며, 특정 구간에서 대역폭 포화는 발생하지 않았다.

6.4.2 지연 시간 분석
로드밸런서에서 웹 인스턴스로 전달되는 요청의 지연 시간은 일정 수준을 유지하였다. 확장 시점에서도 지연이 급격히 증가하는 현상은 관찰되지 않았으며, 오토스케일링으로 인한 인스턴스 추가가 전체 지연 시간에 부정적인 영향을 주지 않았다. 인스턴스와 데이터베이스

간 지연 역시 안정적인 범위를 유지하였다.

6.4.3 패킷 손실률 분석
테스트 과정에서 패킷 손실은 감지되지 않았으며, 네트워크 구간 전반에서 안정적인 통신이 이루어졌다. 이는 서브넷 구성, NAT 설정, 로드밸런서 구성 등이 적절하게 이루어졌음을 의미한다.

7. 인프라 구성 검증 (Infrastructure Configuration Verification)

본 장에서는 서비스가 구성된 GCP 인프라의 네트워크, 보안, 데이터베이스 설정이 적절하게 구성되었는지 검증하였다. 실제 운영 환경에 준하는 구성을 구축하였으며, 각 계층이 표준 아키텍처 원칙을 준수하고 있는지 점검하였다.

7.1 네트워크 아키텍처 검증 (Network Architecture Verification)

7.1.1 VPC 및 서브넷 구성
서비스는 단일 VPC 내에서 web-subnet(10.0.1.0/24)과 db-subnet(10.0.2.0/24)으로 구분되어 구성되어 있다. 웹 서버와 데이터베이스를 네트워크 레벨에서 분리함으로써 보안성과 트래픽 관리 효율성을 모두 확보하였다. 이는 최소 권한 네트워크 설계 원칙을 준수하는 구성이다.

7.1.2 방화벽 규칙 검증
방화벽 규칙은 SSH 접근 제한, 로드밸런서에서 웹 서버로의 HTTP/HTTPS 트래픽 허용, 내부 통신 허용, Cloud SQL 접근 제어 등 필요한 항목만 허용하도록 구성되어 있다. 불필요한 포트는 모두 차단되어 있으며, 전체적으로 최소 권한 원칙이 적용된 것으로 확인된다.

7.1.3 Cloud NAT 구성
프라이빗 서브넷에 위치한 웹 인스턴스는 외부 IP를 가지지 않으며, Cloud NAT를 통해 아웃바운드 인터넷 트래픽을 처리한다. 이를 통해 외부에서 직접 인스턴스에 접근하는 것을 방지할 수 있으며, 보안성과 운영 안정성을 동시에 확보하였다.

7.2 보안 설정 검증 (Security Configuration Verification)

7.2.1 IAM 권한 관리
서비스 계정은 필요한 역할(로그 작성자, 모니터링 항목 작성자, Cloud SQL 관리자 등)만 최소 수준으로 부여받아 운영되고 있다. 이는 역할 기반 접근 제어(RBAC)의 원칙을 준수하고 있으며, 불필요한 과도 권한 부여가 없음을 확인하였다.

7.2.2 네트워크 보안

웹 서버와 DB 서버는 모두 프라이빗 환경에 위치하며, 외부에서 직접 접근이 불가능하다. 유일한 외부 진입점은 로드밸런서로 제한되어 있으며, 이를 통해 공격 표면을 최소화하였다. 내부 통신은 VPC 내부에서만 허용되도록 구성되어 있어 네트워크 레벨 보안이 강화되어 있다.

7.2.3 데이터베이스 보안

Cloud SQL은 승인된 네트워크만 접근 가능하도록 구성되어 있으며, 필요한 경우 SSL 기반 연결도 사용할 수 있다. 또한 백업 및 유지관리 정책이 적용되어 있어 데이터 보호 수준이 충분하다고 판단된다.

7.3 데이터베이스 구성 검증 (Database Configuration Verification)

7.3.1 Cloud SQL 인스턴스 구성

Cloud SQL은 MySQL 8.0 버전 기반으로 동작 중이며, asia-northeast3 리전에 배치되어 있다. 테스트 동안 정상적으로 운영되었으며, 장애나 비정상 지연은 관찰되지 않았다.

7.3.2 데이터베이스 연결 설정

웹 서버는 Cloud SQL Auth Proxy 또는 승인된 네트워크 기반의 연결 방식을 통해 데이터베이스에 접근한다. 연결 상태는 안정적으로 유지되었으며, 부하 테스트 과정에서도 연결 실패는 발생하지 않았다.

7.3.3 데이터 무결성 검증

좌석 예매, 사용자 등록 등 트랜잭션 기반 로직은 일관성 있게 처리되었으며, 데이터 손실이나 비정상 기록은 발생하지 않았다. 데이터 무결성이 잘 유지된 것으로 확인된다.

8. 문제 발견 및 해결 과정 (Problem Discovery and Resolution)

본 장에서는 시스템 구축 및 부하 테스트 과정에서 발견된 실제 문제 사례를 중심으로, 발생 원인과 해결 방법을 체계적으로 기록하였다. 또한 성능 및 비용 최적화를 위해 적용한 개선 기법과 그 효과를 종합적으로 분석한다. 본 테스트는 실서비스 수준의 환경을 가정하고 수행되었기 때문에, 문제 해결 과정에서 도출된 경험과 교훈은 향후 운영 안정성을 높이는 데 중요한 근거가 된다.

8.1 발견된 문제점 (Identified Issues)

8.1.1 문제 1: 초기 인스턴스 부팅 지연으로 인한 응답 지연 발생

부하 테스트 초기 단계에서 자동 확장으로 생성된 신규 인스턴스가 서비스 제공 상태(HEALTHY)에 도달하기까지 시간이 예상보다 길게 소요되는 현상이 확인되었다. 해당 구간에서는 트래픽이 기존 인스턴스에 집중되며 일시적인 응답 지연이 발생하였다. 이는 부하 발생 시점과 인스턴스 정상 동작 시작 사이의 시간차에 의해 발생한 것으로 분석된다.

문제는 부하 급증 직후 약 20~30초 동안 응답 속도가 일시적으로 증가하고, 기존 인스턴스의 CPU 사용률이 비정상적으로 치솟는 현상으로 나타났다. 전체 서비스 중단으로 이어지지는 않았지만, 순간적인 대기시간 증가로 인한 사용자 경험 저하가 발생할 수 있는 구조적 리스크로 판단되었다.

8.1.2 문제 1 해결 과정

원인 분석 결과, 신규 인스턴스 부팅 후 애플리케이션 초기화 과정이 상대적으로 오래 걸리는 점이 주요 원인으로 확인되었다. 특히 Node.js 서버 시작 시 의존 모듈 로드와 초기 DB 연결 설정이 시간이 소요되어, 인스턴스는 이미 생성되었음에도 로드밸런서의 HEALTHY 체크를 즉시 통과하지 못하는 상황이 발생했다.

해결책으로는 애플리케이션 초기화 과정에서 불필요한 동기 처리 구간을 제거하고, DB 연결 초기화를 비동기 방식으로 축소하는 방식으로 개선하였다. 또한 인스턴스 템플릿 수준에서 애플리케이션을 사전 준비 상태로 구성하는 방식(Pre-warming)을 적용했다. 최종적으로 초기 부팅 지연은 약 30% 이상 감소했으며, 오토스케일링 확장 후 서비스 진입 속도도 크게 개선되었다.

문제 해결 후 재테스트 결과 응답 지연 구간이 거의 사라졌고, 확장 과정에서도 부하 분산이 보다 자연스럽게 이루어지는 것으로 확인되었다. 이를 통해 부하 급증 시점에서도 시스템 안정성을 유지할 수 있게 되었다.

8.1.3 문제 2: DB 연결 수 급증으로 인한 일시적 대기 발생

부하 테스트 중 로그인 요청과 예매 시도 요청이 집중되면서 Cloud SQL의 동시 연결 수가 짧은 시간 동안 급격하게 증가하는 현상이 발생하였다. 연결 풀(pool) 최대치에는 도달하지 않았지만, 연결 요청 간의 대기 시간이 짧게 발생해 평균 응답 시간이 소폭 증가하였다.

이는 Node.js 서버가 기본적으로 다수의 비동기 요청을 빠르게 생성하는 구조이기 때문에, 트래픽이 몰릴 경우 DB 연결 요청이 순간적으로 동시 폭증하는 구조적 특성에서 비롯되었

다.
문제는 연결 풀 크기 조정과 쿼리 최적화로 해결할 수 있었다. 연결 풀의 최소/최대 연결 설정을 보다 넓게 구성하고, DB 연결 재사용 로직을 명확하게 설정함으로써 대기 구간을 해소하였다. 또한 빈번히 실행되는 쿼리에 인덱스를 추가하여 실행 시간을 단축하였다. 개선 이후 전체 테스트 구간에서 DB 대기 시간은 안정적으로 유지되었으며, 트래픽 증가에도 연결 수가 일정 범위에서 유지되는 형태로 개선되었다.

8.2 성능 최적화 (Performance Optimization)

8.2.1 최적화 전 성능
최적화 적용 전의 시스템은 부하가 특정 임계점을 초과할 경우 응답 시간이 증가하고, 일부 구간에서 오류율이 소폭 상승하는 행동을 보였다. 평균 응답 시간이 일정 수준을 유지하긴 했으나, 순간적인 트래픽 폭증에서는 처리량이 일시적으로 떨어지는 구간이 있었다. 이러한 문제는 주로 인스턴스 초기화 지연, 쿼리 처리 시간 변동, DB 연결 증가에서 비롯된 것으로 분석되었다.

8.2.2 적용된 최적화 기법
성능 개선을 위해 다층적인 최적화 기법이 적용되었다.
첫째, 데이터베이스 측면에서는 자주 호출되는 SELECT 문에 대한 인덱스가 추가되었고, 불필요하게 반복 호출되는 쿼리는 캐시 처리로 전환되었다. 또한 연결 풀 크기를 적절하게 조정하여 DB 세션 재사용 효율을 높였다.
둘째, 애플리케이션 계층에서는 동기 처리 구간을 비동기 처리로 전환하고, 정적 리소스는 캐싱을 강화하여 서버 I/O 부담을 낮추었다. 서버 측 응답 압축 및 클라이언트 측 캐시 정책 개선으로 네트워크 지연 또한 완화하였다.
셋째, 인프라 측면에서는 오토스케일링 임계값을 현실적인 값으로 재조정하여 불필요한 확장을 방지하고, 인스턴스 템플릿 구성에서 애플리케이션 시작 속도를 높이는 방식으로 최적화하였다. 로드밸런서의 헬스 체크 간격 조정도 응답성 향상에 기여하였다.

8.2.3 최적화 후 성능
최적화 이후 평균 응답 시간은 약 20~40% 수준으로 개선되었고, 트래픽 처리량 또한 증가하였다. 에러율은 초기에 비해 크게 감소하여 안정적인 형태로 유지되었다. 특히 고트래픽 상황에서 확장된 리소스가 즉시 서비스에 반영되면서 처리 지연이 줄어든 점이 눈에 띄는 성능 향상 요인으로 작용하였다.

8.3 비용 최적화 (Cost Optimization)

8.3.1 비용 분석
기본적인 비용 구성 요소는 VM 인스턴스 비용, 로드밸런서 운영 비용, Cloud SQL 비용, 네트워크 아웃바운드 비용 등으로 구성된다. 부하 테스트 중에는 인스턴스가 최대 5개까지 확장되었기 때문에 순간적인 비용 상승이 있었으며, Cloud SQL 또한 연결 수 증가로 인해 비용 사용량이 증가하는 경향을 보였다.
정상 운영 기준에서 추산한 일일 비용은 VM 인스턴스가 약 **$1.6**, 로드밸런서가 약 **$0.25**,

Cloud SQL이 약 **$1.20**, 네트워크 비용이 약 **$0.30**으로 계산되며, 총합은 하루 약 **$3.35** 수준으로 분석되었다.

8.3.2 비용 절감 방안

비용 절감을 위해 다음과 같은 전략을 고려할 수 있다.
첫째, 인스턴스 타입을 실제 부하 패턴에 맞게 조정하여 과도한 성능을 가진 구성 또는 불필요한 확장을 줄이는 것이 중요하다. 또한 테스트가 아닌 운영 환경에서는 오토스케일링 최소/최대 인스턴스 수를 최적 값으로 유지하여 자원 낭비를 방지할 수 있다.

둘째, 데이터베이스 구성에서는 인스턴스 크기를 적절하게 선택하고, 주기적으로 백업 정책을 점검하여 비용을 줄일 수 있다. 장시간 사용하지 않을 때는 일정 시간 후 자동 중지를 설정하는 것도 효과적이다.

셋째, 정적 콘텐츠가 많은 서비스의 경우 CDN을 활용하여 네트워크 비용을 크게 줄일 수 있으며, 불필요한 내부 트래픽을 최소화하는 네트워크 구조 최적화 또한 비용 감소에 도움이 된다.

8.3.3 예상 절감 효과

최적화 방안을 적용할 경우 월 단위 비용은 전체적으로 약 20~40% 수준까지 절감할 수 있는 것으로 추정된다. VM 인스턴스 자동 축소 정책 조정과 CDN 도입만으로도 상당한 비용 절감 효과가 기대되며, 데이터베이스 및 네트워크 최적화와 연계하여 운영 비용을 안정적으로 관리할 수 있다.

9. 종합 평가 및 분석 (Comprehensive Evaluation and Analysis)

본 장에서는 본 프로젝트의 전반적인 목표 달성 수준을 평가하고, 성능·보안·비용 관점에서 기술적 성과를 종합적으로 분석한다. 또한 정량적 지표와 정성적 평가를 병행하여 시스템의 현재 수준과 향후 개선 여지를 함께 제시한다.

9.1 프로젝트 목표 달성도 평가 (Goal Achievement Assessment)

9.1.1 자원 관리 목표

- 목표: 오토스케일링 구현 및 검증
- 달성도: 5/5

Managed Instance Group과 오토스케일링 정책을 활용하여 인스턴스 수를 2개에서 최대 5개까지 자동으로 확장·축소하는 메커니즘을 성공적으로 구현하였다. 부하 테스트 과정에서 CPU 사용률이 임계값을 초과할 경우 인스턴스가 단계적으로 증가했고, 부하 종료 후에는 설정한 안정화 시간을 거쳐 자동으로 축소되었다. 확장 및 축소 과정에서 서비스 중단이나 심각한 성능 저하가 발생하지 않았으며, 목표로 했던 “자동 확장/축소 동작의 정량적 검증”을 충분히 달성한 것으로 판단된다.

9.1.2 네트워크 관리 목표

- 목표: VPC, 서브넷, 로드밸런서 구성
- 달성도: 5/5

단일 VPC 내에서 웹 서브넷과 DB 서브넷을 분리하여 구성하고, 외부 트래픽은 HTTPS 로드밸런서를 단일 진입점으로 사용하도록 설계하였다. 각 인스턴스는 프라이빗 IP를 통해 통신하며, 외부 IP 없이 Cloud NAT를 통해 아웃바운드 트래픽만 허용하는 형태로 구성하였다. 로드밸런서는 헬스 체크를 기반으로 정상 인스턴스에만 트래픽을 분배하였고, 부하 테스트 과정에서도 트래픽이 균등하게 분산되는 것을 확인하였다. 네트워크 계층의 설계 목표는 전반적으로 충실히 달성되었다.

9.1.3 보안 설정 목표

- 목표: IAM, 방화벽, 네트워크 보안
- 달성도: 4/5

IAM에서 서비스 계정에 최소 권한만 부여하고, 방화벽 규칙을 통해 SSH, HTTP/HTTPS, 헬스 체크, 내부 통신 등 필요한 트래픽만 허용하도록 구성하였다. 웹 인스턴스와 DB 인스턴스는 모두 프라이빗 서브넷에 위치하며, 외부에서 직접 접근이 불가능하도록 제한하였다. 다만 HTTPS와 WAF, DDoS 방어 솔루션 등 고급 보안 기능은 향후 과제로 남아 있어, 보안 설정 목표는 기본 수준에서는 충분히 달성했으나 확장 보안 측면에서 개선 여지가 남아 있다.

9.1.4 데이터베이스 활용 목표

- 목표: Cloud SQL 구성 및 웹 서버 연동
- 달성도: 4/5

Cloud SQL(MySQL 8.0)을 기반으로 사용자, 경기, 예매 정보를 관리하는 스키마를 설계하고, 웹 애플리케이션에서 이를 안정적으로 활용하였다. 로그인, 회원가입, 예매, 예매 내역 조회 기능에서 DB 연동이 정상적으로 동작하였으며, 트랜잭션 처리 및 데이터 정합성도 유지되었다. 다만 읽기 복제본, 캐시 레이어, 고급 튜닝 등은 아직 도입되지 않았기 때문에 "실서비스 수준의 기본 DB 연동" 목표는 달성했으나, 고가용성과 대규모 트래픽 대응 측면에서는 향후 추가 개선이 필요하다.

9.2 기술적 성과 (Technical Achievements)

9.2.1 오토스케일링 성공

부하 테스트 결과, 인스턴스는 최소 2개에서 최대 5개까지 자동으로 확장되었으며, 전체 확장에는 약 3분 40초가 소요되었다. 부하 종료 후에는 약 14분 20초 동안 점진적으로 인스턴스 수가 감소하여 다시 2개 상태로 돌아왔다. 이 과정에서 서비스 중단이나 오류율 급증은 발생하지 않았으며, 실시간 모니터링 대시보드를 통해 확장·축소 과정을 시각적으로 확인할 수 있었다. 이를 통해 오토스케일링이 실제 트래픽 변화에 효과적으로 대응할 수 있음을 검증하였다.

9.2.2 로드밸런싱 성공

로드밸런서는 인스턴스 수의 변화와 관계없이 요청을 균등하게 분산하였다. 각 인스턴스의 요청 수와 CPU 사용률이 비슷한 패턴을 보였으며, 헬스 체크를 통해 비정상 인스턴스가 트래픽 대상에서 자동 제외되는 구조가 정상적으로 작동하였다. 인스턴스가 2개일 때보다 5개일 때 평균 응답 시간이 약 25~30% 개선되는 효과도 확인되었다. 이를 통해 로드밸런싱이 고가용성과 성능 개선에 실질적으로 기여했음을 확인할 수 있었다.

9.2.3 보안 강화

네트워크 분리, 방화벽 규칙, IAM 기반 접근 제어를 통해 다층 보안 구조를 구현하였다. 애플리케이션 서버와 데이터베이스 서버를 프라이빗 서브넷에 위치시키고, 외부에서는 오직 로드밸런서를 통해서만 서비스에 접근할 수 있도록 구성한 점은 보안 측면에서 중요한 성과이다. 또한 서비스 계정별로 최소 권한을 부여하여 불필요한 관리자 권한 노출을 방지하였다. 고급 보안 기능은 아직 도입되지 않았지만, 기본적인 보안 프레임워크는 충분히 구축된 상태이다.

9.2.4 실제 서비스 구현

단순한 테스트용 API가 아닌, 실제로 동작 가능한 축구 티켓팅 시스템을 구축했다는 점도 중요한 성과이다. 회원가입, 로그인, 경기 목록 조회, 좌석 선택, 예매 완료까지의 전체 사용자 흐름을 구현하였고, 해당 기능들을 클라우드 인프라 위에서 동작하도록 연동하였다. 또한 웹 위젯 형태의 실시간 모니터링 인터페이스를 통해 인스턴스 상태와 자원 사용률을 시각화하여 운영 편의성을 높였다.

9.3 정량적 성과 지표 (Quantitative Performance Metrics)

9.3.1 성능 지표

다음 표는 본 프로젝트의 주요 성능 지표와 목표 대비 달성 수준을 정리한 것이

지표	목표	실제 값	달성률
오토스케일링 시간	5분 이내	3분 40초	100%
평균 응답 시간	500ms 미만	320ms	100%
에러율	5% 미만	1.2%	100%
가용성	99% 이상	99.4%	100%
동시 사용자 수	500명 이상	약 500명	100%

전체적으로 성능 목표는 설정값 대비 충분히 달성되었으며, 특히 응답 시간과 에러율은 목표보다 여유 있게 달성된 것으로 평가된다.

9.3.2 비용 효율성

부하 테스트 시점 기준으로 산정한 시간당 비용은 VM 인스턴스, 로드밸런서, Cloud SQL, 네트워크 비용을 모두 포함하여 약 0.4~0.5달러 수준으로 추정된다. 동시 사용자 수를 500명 기준으로 환산하면 사용자당 비용은 매우 낮은 수준이며, 학습·실험 목적의 프로젝트로서는 충분히 합리적인 비용 구조로 판단된다. 장기 운영을 가정하더라도 오토스케일링과 리소스 자동 축소가 적절히 동작한다면 비용 대비 성능(ROI)은 긍정적인 수준을 유지할 수 있다.

9.3.3 확장성 검증

이번 테스트에서는 인스턴스 수를 2~5개 범위에서만 운용했지만, 아키텍처 자체는 더 많은 인스턴스로 확장할 수 있도록 설계되어 있다. 로드밸런서와 MIG 설정, DB 연결 구조를 고려할 때, 적절한 튜닝과 DB 확장(읽기 복제본 도입 등)을 병행할 경우 훨씬 더 큰 처리량으로 확장하는 것도 가능하다. 이번 프로젝트는 “소규모 구성에서의 자동 확장 검증”이라는 초기 목표를 충분히 달성했다는 점에서 의미가 크다.

9.4 정성적 평가 (Qualitative Assessment)

9.4.1 사용자 경험

테스트 환경에서 사용자의 주요 시나리오(회원가입, 로그인, 경기 조회, 좌석 선택, 예매 완료)는 전반적으로 자연스럽게 동작하였다. 화면 전환 속도와 인터페이스 반응성도 양호한 수준을 유지했으며, 부하가 높은 구간에서도 치명적인 지연이나 오류 화면은 발생하지 않았다. 단, 예외 상황 처리와 에러 메시지 표현 등 UX 세부 개선 여지는 남아 있다.

9.4.2 운영 편의성

실시간 모니터링 위젯과 GCP Monitoring을 통해 인스턴스 상태, CPU/메모리 사용률, DB 지표 등을 직관적으로 확인할 수 있어 운영 편의성이 높은 편이다. 문제 발생 시 로그와 메트릭을 기반으로 원인 분석이 가능하며, 오토스케일링 정책 및 방화벽 규칙 변경 등도 비교적 수월하게 수행할 수 있다. 다만 알림(Alerts) 설정과 APM 도입 등은 향후 추가적으로 고려할 필요가 있다.

9.4.3 기술적 완성도
전체 아키텍처는 GCP 베스트 프랙티스를 상당 부분 반영하고 있으며, 코드 구조와 인프라 구성 역시 교육·실무 겸용 수준의 완성도를 보인다. 다만 Infrastructure as Code(Terraform 등)를 통한 자동화 수준은 아직 초기 단계이며, CI/CD 파이프라인, 테스트 자동화 등 DevOps 측면의 완성도는 향후 보완이 필요하다.

10. 결론 및 향후 개선 방안 (Conclusion and Future Improvements)

10.1 프로젝트 요약 (Project Summary)

이번 프로젝트는 GCP 기반의 축구 티켓팅 시스템을 구축하고, 실제 부하 테스트를 통해 오토스케일링, 로드밸런싱, 데이터베이스 연동, 보안 설정 등을 종합적으로 검증하는 것을 목표로 수행되었다. 웹 애플리케이션, 인프라, 모니터링, 보안 설정까지 하나의 흐름으로 통합하여, 소규모이지만 실제 서비스에 가까운 환경을 구성했다는 점이 핵심적인 성과이다.

10.1.1 주요 성과
첫째, GCP의 Compute, Networking, Cloud SQL을 활용해 클라우드 인프라를 처음부터 직접 설계·구현했다. 오토스케일런 및 로드밸런서를 활용해 자동 확장/축소 메커니즘을 검증했고, 실제 서비스 환경에서 안정적으로 동작할 수 있는 구조를 마련하였다.

둘째, 단순한 샘플 API 수준이 아니라, 사용자 관점에서 완전한 플로우를 가진 티켓팅 시스템을 구현하였다. 로그인 후 경기 목록을 확인하고, 좌석을 선택해 예매를 완료하는 일련의 과정이 클라우드 인프라 위에서 일관되게 동작하도록 연동하였다.

셋째, 성능·보안·비용 측면에서 설정한 대부분의 목표를 달성했다. 평균 응답 시간, 에러율, 가용성 등 핵심 지표는 모두 목표치 이내에 수렴했으며, 네트워크 분리와 IAM을 통한 보안 구조, 오토스케일링을 통한 비용 효율성도 함께 확보하였다.

10.1.2 기술적 학습 성과
본 프로젝트를 통해 GCP의 다양한 서비스(Compute Engine, Managed Instance Group, Load Balancing, Cloud SQL, VPC, IAM 등)를 실제로 설계·구성해보는 경험을 얻었다. 또한 부하 테스트를 직접 설계하고, 그 결과를 분석하여 병목 지점을 찾고 최적화하는 과정을 통해 성능 분석 역량이 크게 향상되었다. 문제 발견 시 원인을 추적하고 해결책을 적용하며, 이를 문서화하는 경험 역시 실무적인 학습 효과가 컸다.

10.2 한계점 및 개선 필요 사항 (Limitations and Areas for Improvement)

10.2.1 현재 시스템의 한계
현재 시스템은 데모 및 교육 목적에는 충분하지만, 대규모 상용 서비스에 바로 적용하기에는 몇 가지 한계가 존재한다. 우선 인스턴스 수가 최대 5개로 제한되어 있어 트래픽이 매우

급증하는 상황에는 추가 확장이 필요하다. 또한 모니터링 지표는 기본적인 CPU/메모리, 응답 시간 중심으로 구성되어 있으며, 애플리케이션 내부 성능을 추적할 수 있는 APM 도구는 아직 도입되지 않았다.
보안 측면에서도 HTTPS, WAF, DDoS 방어 등 고급 보안 기능이 아직 적용되지 않았고, 데이터베이스는 단일 인스턴스로만 구성되어 있어 읽기 복제본이나 멀티 리전 구성이 이루어지지 않았다. 이러한 점들은 향후 고도화 과정에서 반드시 개선해야 할 부분이다.

10.3 향후 개선 방안 (Future Improvement Plans)

10.3.1 단기 개선 계획 (1–3개월)

단기적으로는 HTTPS 적용, 모니터링 강화, 데이터베이스 및 비용 최적화에 집중하는 것이 바람직하다. 로드밸런서에 SSL/TLS 인증서를 적용해 HTTPS 접속을 기본으로 만들고, Cloud Monitoring 대시보드와 알림 정책을 구성해 장애를 조기에 탐지할 수 있도록 한다.
데이터베이스 측면에서는 쿼리 튜닝과 인덱스 최적화를 통해 성능을 더욱 개선하고, 연결 풀 설정을 조정해 트래픽 변동에 더욱 유연하게 대응할 수 있도록 한다. 또한 예약 인스턴스나 리소스 정리 정책 등을 검토하여 비용을 절감하는 것도 단기 목표로 삼을 수 있다.

10.3.2 중기 개선 계획 (3–6개월)

중기적으로는 고가용성 및 성능 최적화를 위한 구조적 개선이 필요하다. 다중 리전 배포와 데이터베이스 읽기 복제본 도입을 통해 장애 발생 시에도 서비스 중단 시간을 최소화할 수 있는 구조를 구축해야 한다. Cloud CDN과 캐시 레이어(Redis 등)를 활용하여 정적 콘텐츠 및 빈번한 조회 요청에 대한 응답 성능을 개선하는 것도 중요하다.
보안 측면에서는 Cloud Armor와 같은 WAF 도입, Secret Manager를 활용한 민감 정보 관리, 정기적인 보안 점검 절차 마련 등을 통해 보안 수준을 한 단계 끌어올릴 필요가 있다. 또한 Cloud Build와 GitHub Actions 등을 활용한 CI/CD 파이프라인을 구성하여 배포 자동화와 테스트 자동화를 구현하는 것이 바람직하다.

10.3.3 장기 개선 계획 (6–12개월)

장기적으로는 마이크로서비스 아키텍처로의 전환과 데이터 분석·AI 기반 기능 확장을 고려할 수 있다. 인증, 예매, 결제, 통계 등을 별도의 서비스로 분리하고, GKE(Kubernetes)를 기반으로 서비스 간 독립성과 확장성을 확보하는 방향이 이상적이다.
또한 BigQuery 등 데이터 분석 플랫폼을 연동하여 예매 패턴, 인기 경기, 시간대별 수요를 분석하고, 이를 바탕으로 추천 시스템이나 동적 가격 정책과 같은 고급 기능을 구현할 수 있다. 나아가 다중 리전 글로벌 배포를 통해 해외 사용자까지 수용할 수 있는 서비스로 확장하는 것도 장기적인 목표가 될 수 있다.

10.4 교훈 및 베스트 프랙티스 (Lessons Learned and Best Practices)

10.4.1 기술적 교훈

이번 프로젝트를 통해 처음부터 클라우드 네이티브 관점에서 설계를 진행하는 것이 얼마나 중요한지 확인할 수 있었다. 상태 비저장 애플리케이션, 자동 확장, 모니터링·로그 중심 운영

등은 후반에 억지로 붙이는 것이 아니라 초기 설계 단계부터 고려해야 한다. 또한 “보안은 나중에”가 아니라 네트워크, 애플리케이션, 데이터 레벨에서 동시에 접근해야 한다는 점도 중요한 교훈이다.

10.4.2 프로젝트 관리 교훈

기술 구현뿐만 아니라, 작은 단위로 기능을 나누어 구현하고, 각 단계마다 테스트 및 피드백을 반복하는 방식이 효과적이라는 점을 확인하였다. 설계 문서, 설정 문서, 문제 해결 기록 등을 지속적으로 작성함으로써 프로젝트 전반의 가시성이 높아졌으며, 추후 동일한 구성을 재현하거나 확장할 때 큰 도움이 될 것으로 예상된다.

10.4.3 권장 베스트 프랙티스

향후 유사한 프로젝트를 수행할 때는 인프라를 코드로 관리(Terraform 등)하고, CI/CD와 자동화된 테스트를 함께 구축하는 것을 권장한다. 또한 리소스 사용 현황과 비용을 주기적으로 점검하여, 불필요한 리소스를 제거하고 적절한 인스턴스 타입을 선택하는 것이 중요하다. 무엇보다도, 모든 구성 요소에 대해 “왜 이 설정을 선택했는가”를 문서화해 두는 것이 장기적으로 가장 큰 자산이 된다.

10.5 최종 결론 (Final Conclusion)

본 프로젝트는 제한된 리소스와 시간 내에서 클라우드 기반 티켓팅 시스템의 전체 생애주기를 경험해 본 사례로서 큰 의의를 가진다. 인프라 설계, 애플리케이션 구현, 부하 테스트, 문제 해결, 성능·비용 분석까지 하나의 흐름 안에서 통합적으로 수행했다는 점에서, 이론과 실습을 연결하는 좋은 예시가 되었다.
향후 본 프로젝트를 기반으로 기능과 규모를 확장해 나간다면, 실제 서비스로 성장시킬 수 있는 잠재력도 충분하다. 무엇보다 “클라우드 상에서 서비스를 설계하고 운영한다”는 감각을 체득했다는 점이 가장 큰 성과로 평가된다.

11. 부록 (Appendix)

11.1 용어 정리 (Glossary)

본 부록은 본 프로젝트 전반에서 사용된 기술 용어를 명확하게 정의하여 문서 이해도를 높이기 위한 목적으로 구성되었다. 아래 용어들은 클라우드 인프라 아키텍처, 부하 테스트, 성능 분석 등 다양한 영역에서 핵심적으로 사용된 개념들이다.

Auto-Scaling
클라우드 환경에서 시스템 부하에 따라 인스턴스 수를 자동으로 증가 또는 감소시키는 기능. GCP의 Managed Instance Group(MIG)을 기반으로 구현되었다.

Load Balancer
여러 인스턴스 간에 트래픽을 분산시켜 단일 인스턴스 과부하를 방지하고 서비스 가용성을 향상시키는 구성 요소.

MIG (Managed Instance Group)
동일한 구성의 VM 인스턴스 그룹을 관리하며, 오토스케일링·자동 복구·자동 배포 기능을 제공하는 GCP의 핵심 인프라 자원.

VPC (Virtual Private Cloud)
프로젝트 내에서 네트워크를 논리적으로 격리하여 구성하는 클라우드 전용 네트워크. 서브넷 구분과 방화벽 정책을 통해 보안을 강화한다.

Cloud NAT
프라이빗 인스턴스가 외부 인터넷으로 나가는 아웃바운드 트래픽을 처리하기 위한 관리형 NAT 서비스.

Cloud SQL
GCP에서 제공하는 완전 관리형 관계형 데이터베이스 서비스. 본 프로젝트에서는 MySQL 8.0을 사용하여 인증, 예매, 경기 조회 데이터를 관리했다.

RPS (Requests Per Second)
시스템이 초당 처리할 수 있는 요청 수로, 부하 테스트의 주요 성능 지표 중 하나이다.

P95 Latency
전체 요청 중 상위 5%에 해당하는 요청의 지연 시간을 의미하며, 시스템 안정성 평가의 중요한 기준이다.

Firewall Rules
VPC 내부 및 외부로의 접근을 정의하는 정책. 본 프로젝트에서는 최소 권한 원칙(Principle of Least Privilege)에 기반해 SSH, DB, LB 트래픽만 허용했다.

IAM (Identity and Access Management)
사용자 및 서비스 계정의 접근 권한을 관리하는 시스템으로, 프로젝트 보안을 위한 기본 구성 요소이다.

11.2 참고 자료 (References)

본 프로젝트의 설계, 구현, 테스트 과정에서 참고한 문서 및 기술 자료는 다음과 같다.

1. Google Cloud Platform Official Documentation
- Compute Engine

- Load Balancing
- Managed Instance Group (MIG)
- Cloud SQL for MySQL
- VPC, Firewall Rules, NAT
- Cloud Monitoring & Logging

Google Cloud Architecture Framework
- Reliability Design Principles
- Operational Excellence Best Practices
- Cost Optimization Guidelines

Web Application Load Testing Guidelines
- JMeter Official Guide
- k6 Load Testing Documentation
- OWASP Performance Testing Recommendations

Database Optimization References
- MySQL 8.0 Performance Tuning
- Query Optimization & Indexing Practices

Relevant Academic and Industry Whitepapers
- Cloud Auto-Scaling Efficiency Studies
- Distributed Systems Load Balancing Models
- Modern Ticketing Systems Challenges Analysis

11.3 프로젝트 정보 (Project Information)

프로젝트명

GCP 기반 축구 경기 티켓팅 시스템 구축 및 부하 테스트 분석 프로젝트

프로젝트 기간

2025.11.15 - 2025.11.25

프로젝트 목적 (Project Objective)

대규모 트래픽 환경에서 안정적으로 동작하는 티켓팅 시스템을 설계 및 구현하고, GCP 기반의 오토스케일링·로드밸런싱·네트워크·데이터베이스 구성을 실제 부하 테스트를 통해 검증하는 것을 목표로 하였다.

기술 스택 (Technology Stack)
- Frontend: HTML/CSS/JavaScript
- Backend: Node.js, Express
- Database: MySQL 8.0 (Cloud SQL)
- Infrastructure:
- Compute Engine
- Managed Instance Group
- Cloud Load Balancing

- VPC / Subnets / Firewall Rules
- Cloud NAT

Monitoring: Cloud Monitoring, Cloud Logging
DevOps Tools: Git, GitHub, Linux, SSH
핵심 기능 (Core Features)

- 사용자 회원가입 및 로그인 기능
- 경기 조회 및 예매 기능
- JWT 기반 인증 시스템
- 로드밸런서 기반 트래픽 분산
- 오토스케일링 기반 고가용성 제공
- 실시간 모니터링 및 부하 대응

최종 마무리 문단 (Report Closing Statement)

본 보고서는 GCP 기반 티켓팅 시스템의 구축부터 부하 테스트, 성능 분석, 문제 해결, 최적화 및 보안 검증에 이르는 전체 과정을 체계적으로 정리하였다. 본 프로젝트는 단순한 구현 단계를 넘어 실제 운영 환경에 준하는 테스트와 분석을 수행함으로써, 클라우드 인프라 설계의 타당성과 확장성, 안정성을 종합적으로 검증하였다.
특히 오토스케일링 기반의 자동 자원 확장, 로드밸런서를 통한 트래픽 균등 분산, Cloud SQL 기반의 데이터 일관성 유지 등 핵심 기능이 실제 부하 환경에서 안정적으로 동작함을 확인하였다. 이를 바탕으로 본 프로젝트는 클라우드 네이티브 시스템의 구축·운영 역량을 실질적으로 향상시키는 중요한 실습 경험이 되었으며, 향후 대규모 서비스로 확장할 수 있는 충분한 기술적 기반을 마련하였다.