**Pusan National University Computer Science and Engineering Technical Report 2023-24**

**클라우드 기반 크레딧 거래 플랫폼**



201724425김민종

201606151김민승

201723148송재원

지도교수 김원석 교수님

목 차

[1. 서론 1](#_Toc148303766)

[1.1. 과제 배경 및 필요성 1](#_Toc148303767)

[1.2. 과제 목표 2](#_Toc148303768)

[1.3. 구현 내용 2](#_Toc148303769)

[2. 과제 설계 3](#_Toc148303770)

[2.1. 서버 설계 3](#_Toc148303771)

[2.2. 클라이언트 설계 7](#_Toc148303774)

[2.3. 배포 파이프라인 설계 9](#_Toc148303778)

[3. 과제 구현 11](#_Toc148303782)

[3.1. 서버 11](#_Toc148303783)

[3.2. 클라이언트 14](#_Toc148303787)

[3.3. K8s 클러스터 18](#_Toc148303791)

[4. 과제 결과 22](#_Toc148303797)

[4.1. K6와 InfluxDB를 이용한 부하 테스트 22](#_Toc148303798)

[4.2. Zipkin을 이용한 트랜잭션 흐름 모니터링 28](#_Toc148303799)

[5. 개선사항 28](#_Toc148303800)

[6. 구성원별 역할 및 개발 일정 29](#_Toc148303801)

[7. 참고 문헌 30](#_Toc148303802)

# 서론

## 과제 배경 및 필요성

2012년 이후 스마트폰 보급률 확산, 무선 통신 인프라와 전자 상거래의 발전으로 모바일 결제 수단이 세계적으로 발전하는 추세이다. 한국 역시 2015년 이후 모바일 인증서 규제 완화를 통해 모바일 결제 서비스가 급속히 성장하고 있고 2022년도 기준 전체 결제의 59%가 모바일 결제로 이루어지고 있다. 그러나 우리나라에서 행해지는 전자 결제 방식들은 대금의 수취인, 수령인 및 거래 대상, 금액의 정보만을 가지고 금전과 재화 및 서비스의 교환이 이루어진다.

우리는 이 결제라는 행위에서 단순히 금전이 오가는 것만이 아니라 부가 이득을 창출할 수 있도록 확장성이 유연한 독자적인 결제 플랫폼을 구축하고자 한다. 예를 들면, 결제 정보로부터 어떤 소비자층이 주로 어떤 재화, 서비스를 구매하는지, 어느 정도의 가격대에서 자주 결제가 이루어지는지 등의 데이터를 수집하는 알고리즘에서부터, 데이터를 바탕으로 사용자 맞춤형 정보를 제공하거나, 사용자 결제 데이터를 바탕으로 사용자의 결제내역, 신용카드 연체 여부, 각종 요금 납부 상황, 모바일 결제 내역, 재테크 상품 가입 현황을 제공하는 등 부가적인 기능을 언제든지 추가 가능한 플랫폼을 개발하는 것을 목표로 한다.

시스템의 확장성 확보를 용이하게 하기 위해 MSA(Microservice Architecture)의 형태로 구현하되 클라우드 컴퓨팅을 활용한다. MSA란 소프트웨어 아키텍처 패턴 중 하나로, 복잡한 애플리케이션을 작은 독립적인 서비스 단위로 분할하여 구성하는 것이다. 각각의 서비스는 자체적으로 독립적인 업무를 수행하며, 필요에 따라 서로 통신하여 전체적인 시스템을 완성하는 아키텍처 패턴이다. 이 독립적인 서비스들을 하나의 모듈로 간주하고 본 시스템의 핵심적인 모듈에 부차적으로 다른 모듈들을 덧붙여 기능을 추가하는 방식으로 커스터마이징 하듯이 시스템을 구성할 수 있게끔 만들어 확장성을 확보할 수 있게 된다.

클라우드 컴퓨팅(Cloud Computing)의 반대되는 개념은 온프레미스(On-premises) 방식인데, 본 MSA 형태의 서비스를 구현하면서 온프레미스 방식보다 클라우드 컴퓨팅 방식을 도입하는 것에 유리한 점이 있다. 온프레미스 방식은 컴퓨터 시스템이나 소프트웨어를 사용자 자신이 보유하고 운영하는 방식을 말한다. 즉, 사용자는 해당 시스템이나 소프트웨어를 직접 구매 및 설치하고, 서버를 직접 관리하고 유지보수를 수행해야 한다. 그러므로 인프라를 확장하기 위해서는 서버를 추가로 구매하고 설치해야 하므로 확장성이 부족하고 더 나아가 보안 측면에서도 사용자가 직접 보안에 대한 책임을 지고 대비해야 하므로 노하우가 부족하다면 보안 취약점이 발생할 수 있다.

또한 유지보수에서도 인력이 많이 들 수 있는 등 부가적인 단점이 따른다. 그에 반해 클라우드 컴퓨팅은 인터넷을 통해 컴퓨팅 자원을 제공하는 기술을 말한다. 즉, 사용자가 인터넷을 통해 필요한 컴퓨팅 자원을 빌려서 사용하는 것이다. 이를 통해 사용자가 직접 인프라나 소프트웨어를 보유하지 않아도 되며, 필요한 만큼 컴퓨팅 자원을 확장하거나 축소할 수 있다. 이를 통해 사용자는 비용을 절감하면서도 필요에 따라 유연하게 자원을 사용할 수 있다. 또한 보안 측면에서도 클라우드 제공업체가 보안에 대한 책임을 지고 있으므로 보안성이 보장된다. 데이터 백업과 같은 보안 기능을 제공하므로 데이터 손실을 예방할 수 있다.

즉 온프레미스 방식을 택하는 것보다 클라우드 컴퓨팅을 도입하면 자원 확장 및 축소, 보안에 대한 부담, 지리적 제약으로부터의 자유도에 있어서 장점을 가지며, 무엇보다도 중요한 점은 자원 확장 및 축소에 용이하다는 점이 MSA의 작은 독립적인 서비스들을 모듈처럼 탈 부착하여 하나의 큰 시스템을 이룬다는 특징과 궁합이 잘 맞아 클라우드 컴퓨팅을 도입하여 구현한다.

본 플랫폼을 개발함으로써 단순히 결제만 하는 것이 아니라, 결제라는 행위에 따른 다른 부가적인 이득을 창출할 가능성을 열어 두고, 확보한 확장성을 바탕으로 다양한 서비스 방식을 실현할 수 있다. 이런 특장점을 들어 개발 필요성을 제고하는 바이다.

## 과제 목표

본 졸업과제는 클라우드 기반 크레딧 거래 플랫폼을 개발하는 것을 목표로 한다. 거래 방식의 확장성을 위해 인증 서비스, 크레딧 서비스를 중심으로 한 MSA를 적용한다. 각각의 서비스는 컨테이너 환경에서 동작하며, 운영 및 확장을 위해 컨테이너 오케스트레이션 기술인 Kubernetes(이하 k8s)를 사용하여 운영한다.

## 구현 내용

본 과제의 목표인 ‘클라우드 환경의 크레딧 거래 플랫폼’을 구현하기 위해 가게의 키오스크 QR 결제를 예시 서비스로 하여 시스템을 구현하되, MSA를 적용하여 다른 거래 방식을 적용할 수 있는 확장성을 부여한다. 그리고 모든 서비스는 자유로운 scale-out을 위해 무상태로 구현되어야 한다. 사용자는 웹서비스를 이용하여 크레딧을 충전하며, 모바일 앱을 통해 QR을 생성하여 인증을 통해 크레딧 거래가 가능하다. 또한 키오스크는 웹 앱으로 구현하고 QR 리더기를 통해 QR 정보를 획득할 수 있어야 한다. 시스템 전체에 대한 요구사항을 정리하면 아래와 같다.

* 플랫폼은 MSA 형태로 클라우드 환경에 구현되어 서비스의 확장이 가능해야 한다.
* 각 서비스는 무상태로 운영되어 자유로운 scale-out이 가능해야 한다.
* 사용자는 웹을 통해 크레딧의 충전이 가능하며, 모바일 앱을 통해 QR 인증 및 결제가 가능해야 한다.
* 키오스크는 웹 앱 형태로 구현하고 QR 리더기를 통해 QR 정보를 획득 및 전송할 수 있어야 한다.

# 과제 설계

## 서버 설계

### 전체 시스템 맥락도

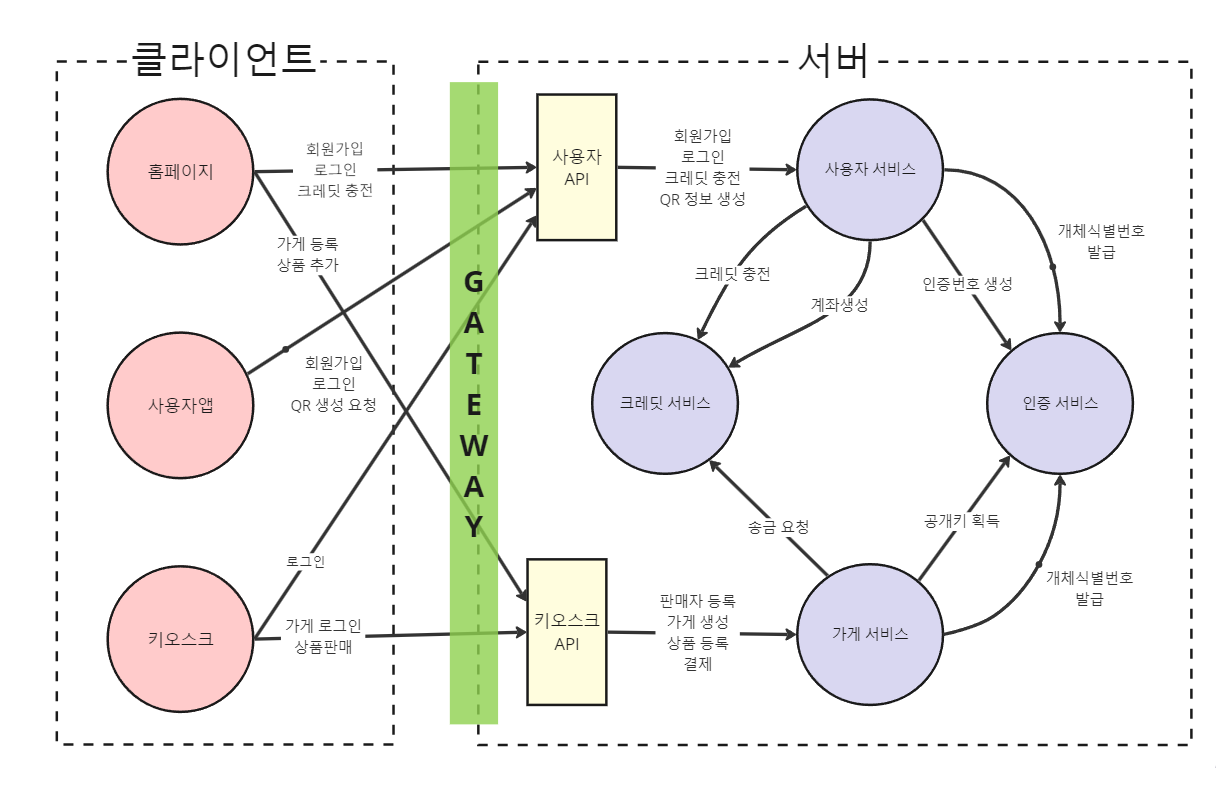


그림 1 전체 시스템 맥락도

전체 시스템은 3가지의 클라이언트와 4가지의 서버 애플리케이션으로 구성되어 있다. 4가지 서버 애플리케이션과 제공하는 대표 API에 대해 소개하겠다.

1. 인증 서버

크레딧 거래 플랫폼의 핵심 서버 애플리케이션으로 플랫폼 이용자 식별번호의 생성과 검증을 담당한다. 대표적으로 제공하는 API는 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 기능 | 엔드포인트 | 외부공개 |
| 식별번호 생성 | 서비스 전체에서 유일한 식별번호 생성 | POST  /serial-number | X |
| 공개키 제공 | 교환 토큰 검증에 필요한 공개키 제공 | GET  /public-key | X |
| 식별번호 교환 토큰 생성 | 식별번호 교환에 사용되는 교환 토큰 생성 | GET  /serial-number/식별번호/exchange | X |

표 1 인증 서버 대표 API

1. 크레딧 서버

플랫폼 이용자의 크레딧을 관리하는 서비스로 계좌의 생성, 크레딧 충전 및 거래를 담당한다.

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 기능 | 엔드포인트 | 외부공개 |
| 계좌생성 | 이름, 아이디, 비밀번호, 이메일을 입력받아 사용자 생성 | POST  /account/계좌이름 | X |
| 크레딧 충전 | 아이디, 비밀번호를 입력받아 jwt 토큰 발행 | POST  /account/계좌이름/load | X |
| 송금 | 사용자 식별번호가 담긴 QR 정보 생성 | POST  /account/보내는계좌/transfer/받는계좌 | X |

표 2 크레딧 서버 대표 API

1. 사용자 서버

사용자를 생성 및 관리하며 크레딧 충전을 요청하거나 결제를 위한 QR 정보를 생성하는 등 사용자와 다른 서버의 중간자 역할을 수행한다.

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 기능 | 엔드포인트 | 외부공개 |
| 사용자  회원가입 | 이름, 아이디, 비밀번호, 이메일을 입력 받아 사용자 생성 | POST  /api/user/signup | O |
| 사용자  로그인 | 아이디, 비밀번호를 입력 받아 사용자 jwt 발행 | POST  /api/login | O |
| QR 정보 생성 | 사용자 식별번호가 담긴 QR 정보 생성 | GET  /api/qr | O |
| 크레딧 충전 | 사용자의 크레딧 충전 | POST  /api/credit | O |

표 3 사용자 서버 대표 API

1. 가게 서버

상품을 판매하고자 하는 가게를 등록 및 관리하며 키오스크를 통한 결제 처리를 담당한다.

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 기능 | 엔드포인트 | 외부공개 |
| 가게 등록 | 가게 이름을 입력 받아 가게 생성 | POST  /api/store | O |
| 가게 로그인 | 사용자 토큰을 이용해 가게 jwt를 발행 | POST  /api/가게번호/login | O |
| 상품 추가 | 상품 이름과 가격을 입력 받아 상품 생성 | GET  /api/가게번호/product | O |
| 결제 | 인식한 QR 코드 정보와 판매 정보를 받아 결제 처리 | POST  /api/가게번호/product/sale | O |

표 4 가게 서버 대표 API

### 보안 설계

* 게이트웨이를 이용한 API 접근

게이트웨이는 클라이언트에게 다양한 서비스에 대한 단일 진입점을 제공한다. 즉, 게이트웨이를 통해 외부에서 접근이 가능한 API를 지정해 줄 수 있다. 서버 애플리케이션은 내부 API와 외부 API를 구분해서 제공하며, 외부 API만이 게이트웨이를 통해 공개된다. 이를 통해 악의적인 사용자의 내부 API 사용을 방지하였다.

* JWT 탈취 방지

본 프로젝트에는 API를 사용하는 두 종류의 사용자가 있다. 바로 일반적인 사용자와 물품 판매에 이용되는 키오스크이다. 우리는 사용자를 구분하기 위해 JWT 인증방식을 사용했다. JWT 인증방식은 탈취에 취약하기 때문에 이를 보완하는 정보를 토큰이나 서버에서 저장해야 한다. 그러나 서버는 Autoscaling을 위해 무상태로 구현되어야 하므로 세션에 저장하는 것은 불가능하고, DB에 저장하는 것 또한 빈번한 호출이 예상되므로 병목현상이 발생할 수 있다. 그래서 우리는 데이터를 토큰 안에 저장할 수 있다는 JWT의 장점을 이용하여 JWT에 탈취를 방지하는 정보를 저장하기로 했다.

먼저 일반적인 사용자의 JWT에는 탈취의 위험을 최소화하는 간단한 방식으로 만료시간 정보를 만들어 30분으로 짧게 설정하였다. 그러나 키오스크는 판매를 위해 긴 시간동안 로그인을 유지해야 하므로 이 방식을 적용할 수 없었다. 그래서 우리는 키오스크의 네트워크는 잘 변하지 않는다는 점을 이용해 키오스크 JWT 발행 시에 IP 정보를 넣었다. 그리고 JWT 검증 시마다 접속 IP와 토큰 속 IP를 비교하여 탈취를 방지하였다.

* QR 위·변조 방지

QR 코드는 외부에 공개되는 정보로 쉽게 위·변조가 가능하다. QR 코드의 사용 목적은 결제의 인증인데 이를 위 변조할 수 있다면 서비스에 큰 지장을 초래할 것이다. 그래서 우리 팀은 식별번호 생성과 서버 간 인증을 중앙화하여 일관된 방식의 보안을 수행할 수 있도록 하고 비대칭키를 이용한 암호화와 서명을 통해 위·변조를 방지하였다. 그림 2는 결제 시 서버별로 처리되는 절차를 순서대로 표현한 것이다.

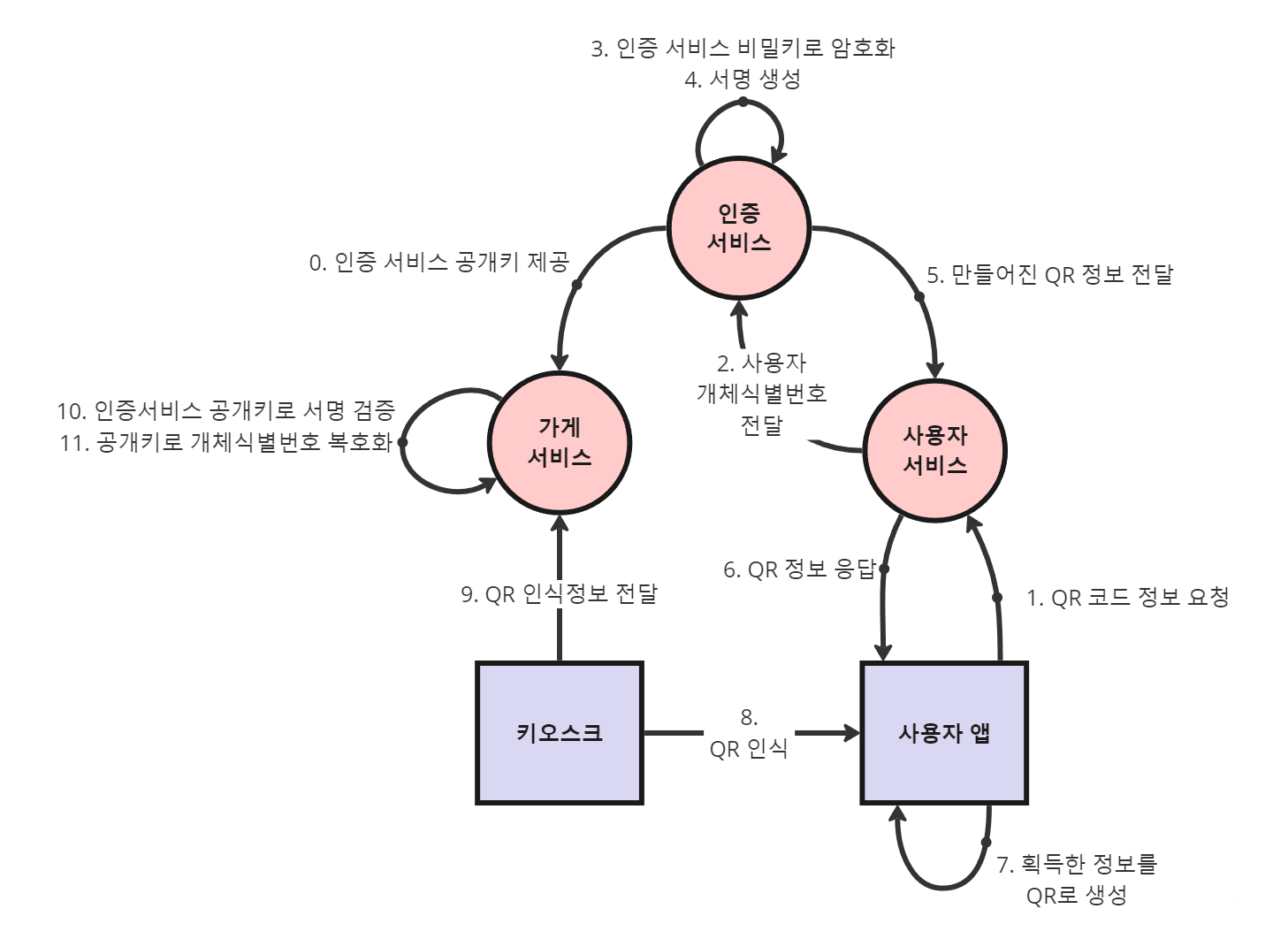


그림 2 결제 시나리오

## 클라이언트 설계

### 홈페이지

홈페이지는 사용자의 회원가입과 크레딧 충전, 가게 서비스 이용자의 가게 등록 및 상품 추가를 담당한다. 간단하게 구현하기 위해 부트스트랩을 이용하여 개발하였다.

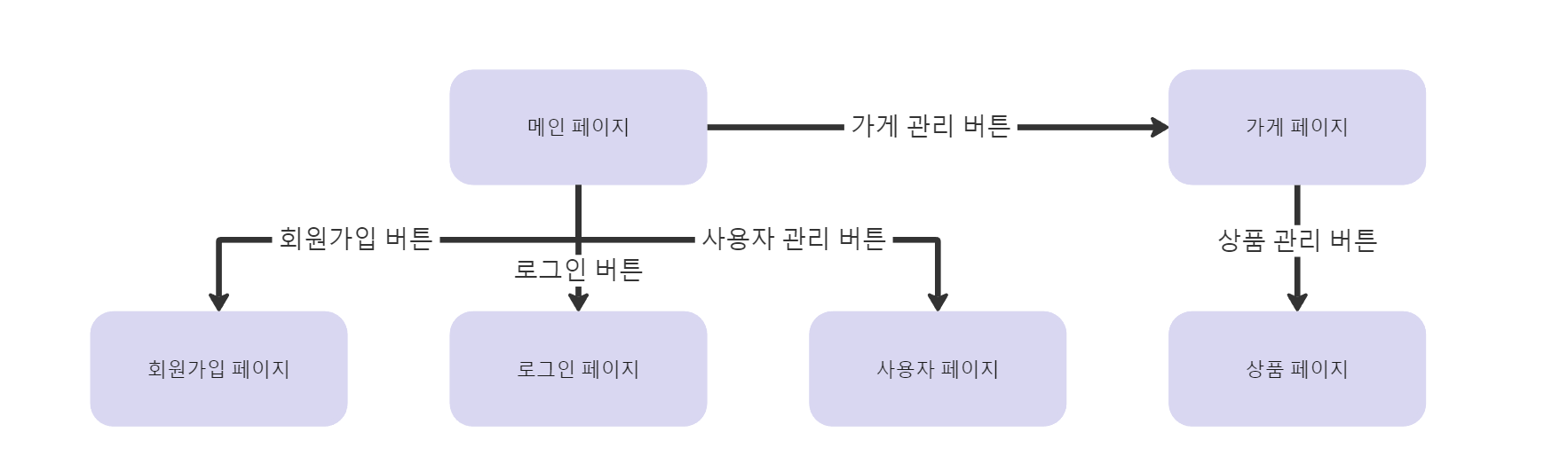


그림 3 홈페이지 화면 시나리오

### 키오스크 앱

키오스크 앱은 상품의 판매 및 결제를 담당한다. 데스크톱용 앱으로 제작하기 위해 일렉트론을 사용하여 개발한다. 그림 4는 키오스크 앱의 화면 시나리오이다.

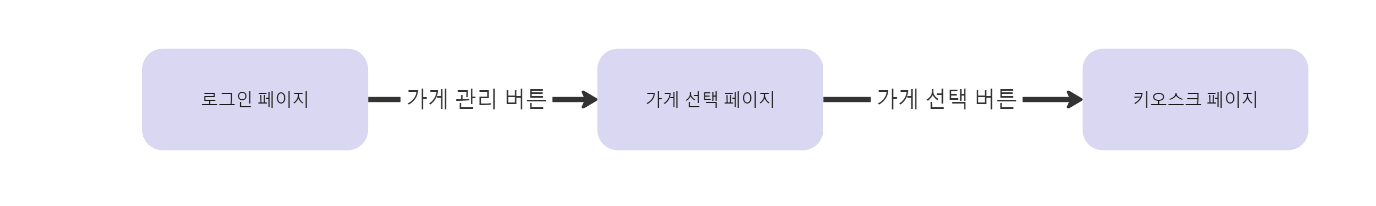


그림 4 키오스크 앱 시나리오

### 사용자 앱

APP을 개발하며 구현해야 할 것은 아래와 같다.

1. 서버와의 통신: 클라이언트 입장에서 로그인, 회원가입, 결제, 충전 등의 기능을 이용하기 위해서는 서버와 통신해야 하는 것은 자명하다. 또한 편의성을 위해 동기적인 처리 방식이 아닌 비동기적 처리 방식으로 수행되게끔 만들어야 한다.
2. 로그인 세션 유지: 한 번 로그인하면 APP에서 제공하는 기능을 이용하기 위해 액티비티를 전환하면서도 로그인된 상태를 유지해야 할 필요가 있다.
3. 문자열 QR 변환: 결제 기능 이용 시, 서버에서 받아온 문자열을 QR로 변환해야 할 필요가 있다.

그림 3은 사용자 앱의 화면 시나리오를 설계한 것이다.

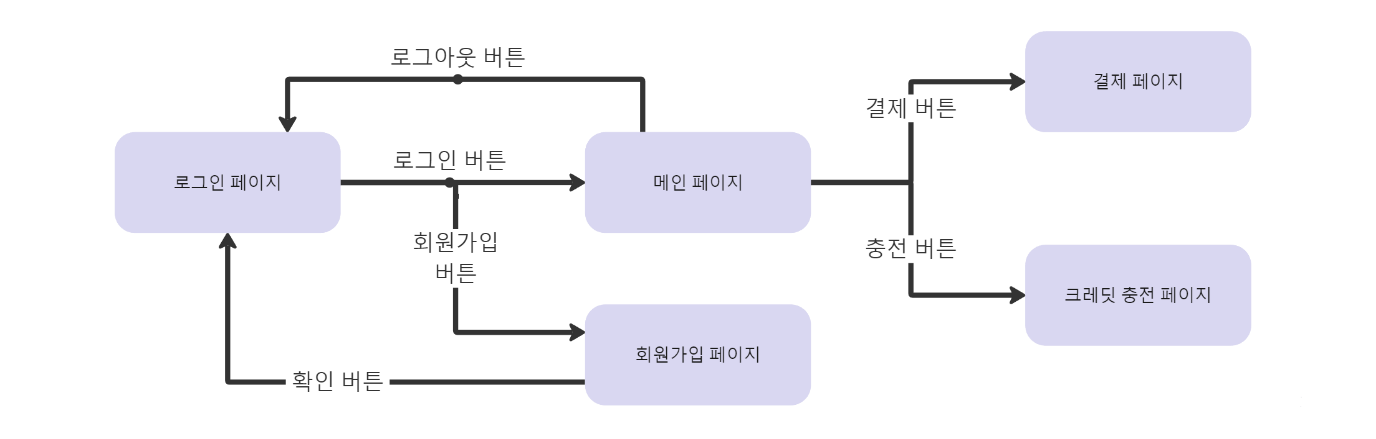


그림 5 사용자 앱 시나리오

## 배포 파이프라인 설계

본 프로젝트가 목표로 하는 확장성과 유연성을 달성하려면 빈번한 주기의 변경과 배포가 필연적으로 발생한다. 이에 변경이 발생하면 자동으로 운영환경에 배포하는 CI/CD 배포 파이프라인을 구성하였다. 또한 파이프라인의 유지보수성을 높이기 위해 Git repository의 코드와 배포 환경이 일치하도록 유지하는 GitOps를 설계에 적용하였다.

텍스트, 도표, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 6 배포 파이프라인 순서도

파이프라인을 구성하고 있는 것을 살펴보면 소스코드나 설정파일을 저장하는 git repository, 프로젝트 이미지가 저장되는 image registry, CI/CD를 가능케 해주는 도구인 Jenkins와 ArgoCD가 존재한다.

### Git repository

파이프라인에는 두 가지의 git repository가 존재한다. 바로 Application repository와 Manifest Repository이다. Application repository는 애플리케이션마다 하나씩 가지고 있으며 개발자가 프로젝트를 저장 및 공유하는 일반적인 git repository이다. Manifest repository는 GitOps 방법론을 구현하기 위한 repository로 k8s 클러스터에 배포되는 애플리케이션 리소스를 정의하는 역할을 담당한다. GitOps를 구현하기 위한 도구인 ArgoCD가 이 repository의 변경을 감지하여 k8s 클러스터에 반영한다.

### Jenkins

Jenkins는 Master Node와 Agent Node로 구성되어 있으며, Master node가 Jenkins Pipeline으로 정의된 모든 흐름을 관장하는 컨트롤 서버의 역할을 담당하고 Jenkins Agent에 일을 분배한다. 우리는 k8s 클러스터 내에 Jenkins를 구성하고, 작업이 발생했을 때 동적으로 Jenkins Agent Pod를 생성하여 Job을 수행하는 형태로 만들었다.

본 프로젝트에서 정한 Jenkins의 빌드 순서는 다음과 같다.

1. 특정 서버 프로그램의 repository에 commit이 발생한다.
2. Jenkins Master Node가 github SCM polling을 통해 commit을 감지하고 Agent를 생성한다.
3. Agent가 commit이 발생한 서버 프로그램이 올라가 있는 repository를 clone한다.
4. Agent가 Jib를 이용하여 서버 프로그램을 이미지로 만들고 docker hub에 업로드한다.
5. Agent가 프로젝트의 manifest repository를 clone한다.
6. Agent가 manifest에 있는 해당 서버의 이미지 버전을 변경한 후 push 한다.

그림 3은 Jenkins가 관리하는 프로젝트의 목록이다. 네 개의 서버 애플리케이션인 인증 서버, 크레딧 서버, 사용자 서버, 가게 서버뿐만 아니라 홈페이지의 HTML 요청을 처리하는 웹서버, 외부 트래픽을 처리하는 게이트웨이, 서비스의 탐색을 담당하는 디스커버리 서버까지 빌드를 처리한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

그림 7 Jenkins로 관리되는 애플리케이션 목록

### ArgoCD

ArgoCD는 GitOps를 구현하기 위한 도구 중 하나로 Manifest repository의 변경 사항을 자동으로 감지하여 k8s 클러스터의 배포 애플리케이션을 자동으로 업데이트할 수 있으며 에러가 발생했을 때 rollback이 손쉽게 가능하다. 또한 프로젝트의 배포 상태나 파드의 로그를 확인할 수 있어 모니터링의 기능도 수행한다. 그림 8은 ArgoCD로 관리하는 사용자 서버의 모습이다. 이미지의 위쪽에서 바라보고 있는 git repository와의 동기화 상태를 확인할 수 있고, 그 아래에 현재 클러스터에 배포된 상태가 알아보기 쉽게 표현되어 있다.

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

그림 8 ArgoCD를 통해 관리 및 모니터링되고 있는 사용자 서비스

# 과제 구현

## 서버

### 게이트웨이

MSA를 구축하면서, 클라이언트 요청의 모니터링, 보안 강화, 서비스 동적 확인이 중요한 요구사항으로 떠올랐다. 이에 API 게이트웨이의 필요성이 대두되었고, Spring Cloud Gateway를 도입하여 이러한 요구사항을 충족하고자 하였다.

본 프로젝트에서 게이트웨이의 기능은 다음과 같다.

1. 로깅: 모든 클라이언트 요청과 서비스 응답을 로그로 기록하여 모니터링과 디버깅을 용이하게 하였다.
2. 보안 헤더: 보안을 강화하기 위해 HTTP해더에 추가 정보를 포함했다. 이는 내부 서비스가 게이트웨이를 거처지 않으면 통신이 불가능하도록 하기 위함이다.
3. 라우팅: 요청을 적절한 서비스로 라우팅하여 효율적인 서비스 분배를 지원한다.

### 문서화

본 프로젝트는 3명의 팀원이 협력하여 진행되었으며, 각자의 역할을 명확하게 나누어 작업하였다. 이에 따라 개발 과정에서 원활한 커뮤니케이션을 위해 문서화가 중요한 역할을 하였다. 따라서 다음과 같은 도구와 기술을 이용하여 개발 및 협업 과정을 간소화하고 효율적으로 관리하였다.

1. [Swagger-UI](http://api.teamcloudnine.link/api-docs)

텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

그림 9 Swagger 중 사용자 서비스 문서

API 문서화를 위해 Swagger-UI를 활용하였다. 이를 통해 API 엔드포인트, 요청 및 응답 형식, 인증 방법 등을 자세하게 문서화하고 다른 팀원에게 API 명세를 시각적으로 제공하였다.

1. [Javadoc](https://koust6u.github.io/user-service-docs/)

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

그림 10 프로젝트 Javadoc 중 사용자 서비스의 jwt 패키지 문서

Java 개발에서는 Javadoc을 활용하여 코드를 문서화하였다. 이를 통해 클래스, 메서드, 필드 등에 대한 역할과 기능을 제공하였다.

1. [에러 코드 정의](https://github.com/C1oud-Nine/c9Pay-templates/blob/main/error-code.md)

|  |  |  |
| --- | --- | --- |
| 에러 코드 | 설명 | 상태 코드 |
| 100 | 회원 아이디 중복 시 | 400 |
| 101 | 로그인이 필요한 경우 | 400 |
| 102 | 존재하지 않은 회원에 대한 요청을 한 경우 | 400 |
| 103 | 로그인 시 회원 비밀번호가 틀린 경우 | 400 |
| 104 | 필요한 정보를 모두 입력하지 않거나 옳지 않은 경우 | 400 |
| 199 | 기타 서버 오류 | 500 |

표 5 사용자 서비스 에러 코드

프로젝트에서 발생할 수 있는 에러 코드를 정의하고, 각 에러 코드에 대한 간단한 설명과 쓰임새를 문서로 작성하였다.

### 장애 대비/대응

프로젝트에서는 MSA의 특징으로 다수의 서비스들 간 통신이 빈번히 발생한다. 이와 같은 환경에서는 하나의 서비스에서 장애가 발생할 경우, 의존 관계에 있는 다른 서비스들로 장애가 전파될 수 있는 문제가 발생한다.

이러한 문제를 극복하기 위해, 본 과제에서는 Resilience4J 오픈소스 라이브러리를 적극적으로 활용하였다. 특히, OpenFeign을 통한 클라이언트와의 통신에서 Resilience4J를 적용하여 다음과 같은 기능을 추가하였다.

서킷 브레이커를 적용하여 장애 시 다른 내부 서비스로의 요청을 일시적으로 차단하고 임시 데이터를 전송 또는 클라이언트에 장애를 알렸다. 또한 초당 3회 이상의 동일한 요청을 비정상적인 요청으로 인지하고 요청을 일시적으로 차단하는 Rate Limiter 기능 추가함으로써 서비스 간 통신 문제와 부하에 대한 안정성을 확보하였다.

또 쓰레드 풀 등의 리소스 부족으로 3초 이상으로 응답이 지연이 발생하는 경우 역시 Time Limiter를 적용하여 요청을 일시적으로 차단하여 불필요한 대기 상태를 방지해 주었다.

## 클라이언트

### 홈페이지

홈페이지는 사용자의 회원가입과 크레딧 충전, 가게 서비스 이용자의 가게 등록 및 상품 추가를 담당한다. 디자인은 PayPal 홈페이지의 디자인을 참고하였고 Bootstrap을 활용하여 비교적 간단하게 구현하였다. 또한 사용자의 구분을 위해 사용자 jwt를 사용하는데 이를 쿠키를 이용하여 관리하였다. 그림11은 홈페이지의 메인이다.

텍스트, 스크린샷, 폰트, 그래픽이(가) 표시된 사진

자동 생성된 설명

그림 11 C9Pay 홈페이지 메인

### 키오스크

키오스크는 상품의 판매를 수행하는 클라이언트이다. 간단하게 데스크톱 앱으로 구현하기 위해 일렉트론을 사용했다. 일렉트론은 Chromium과 Node.js를 이용하여 쉽게 데스크톱 앱을 만들 수 있도록 도와주는 도구이다. 또한 QR 코드 리더기를 연동하여 실제로 QR 코드를 인식할 수 있도록 만들었다. 그림 12는 키오스크에서 상품을 선택하는 페이지이다. 상품을 선택하면 장바구니에 선택한 상품이 담기고 결제하기 버튼을 누르면 QR 코드를 인식하라는 메시지가 출력된다.

텍스트, 스낵, 스크린샷, 음식이(가) 표시된 사진

자동 생성된 설명

그림 12 키오스크 상품 선택 페이지

### 사용자 앱

#### 화면 흐름도 구성

사용자가 서비스를 통해 로그인, 회원가입, 결제, 충전 등의 기능을 이용하기 위해 화면 출력과 입력을 어떤 순서로 받아야 할지 먼저 구성할 필요가 있다. 본 APP의 화면 시나리오는 그림 13과 같다.

텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

그림 13 사용자 앱 화면 흐름도

위와 같이 구성한 화면 흐름도대로 APP의 액티비티 및 레이아웃을 구성한 후, 각 액티비티에 필요한 기능들을 구현해 넣는 방식으로 APP을 개발했다. 그리고 각 액티비티가 전환될 시 기존의 액티비티가 Stack되지 않고 필요 없는 액티비티는 종료되게끔 만들어 최적화하였다.

#### 서버 통신

본 APP은 프로젝트 백엔드 측의 사용자 서비스 API를 활용해 클라이언트 측에 필요한 기능들을 제공한다. 그렇게 하기 위해서는 APP과 API간의 통신이 불가피하다. API와의 통신을 위해 대표적인 REST API 통신 라이브러리인 ‘Retrofit2’를 활용했다.

사전에 백엔드 측과 약속한 API 명세에 따라 데이터 전송을 위한 DTO(Data Transfer Object)를 정의했다. 그 후, http 통신을 구현하기 위해, Retrofit2 라이브러리에서 제공하는 ‘Call’ 인터페이스를 따르는 메소드들을 작성했다. 이를 위해 @HEADER, @POST, @GET 등의 어노테이션들과 앞서 작성한 DTO들을 활용하여, API 호출을 수행하는 메소드들을 각 API 명세에 맞추어 구현했다. 즉, 각 API 호출마다 그에 대응하는 Call 인터페이스를 따르는 메소드들이 일대일방식으로 구현되었다.

코드를 모듈화하고 유지보수를 향상시키며, 중복 코드를 최소화하기 위해 위에서 구현한 메소드들과 Retrofit 객체를 생성하는 메소드를 포함하는 상위의 인터페이스를 도입했다. 따라서, 서버 통신이 필요할 때마다 이 상위의 인터페이스를 사용하여 Retrofit 객체를 생성할 수 있게 했다. 그리고 Retrofit2의 ‘enqueue’ 메소드로 비동기적으로 서버와 통신할 수 있도록 했다.

#### 로그인 세션 유지

APP에 한 번 로그인을 한 후에는 하나의 세션으로 간주하여 APP을 종료하거나 별도로 로그아웃하기 전까지는 로그인 상태를 유지해야 할 필요가 있다. 이를 위해 Cookie나 SharedPreferences, Intent를 활용한 방법을 생각해 보았는데 이 중 Intent를 활용하였다.

로그인 시 회원 인증key를 서버로부터 받아오면 액티비티가 전환되더라도 이 인증key가 유지되어야 한다. 액티비티를 Intent를 통해 전환하면서 그 Intent에 인증key를 포함하는 방식으로 로그인 세션을 유지토록 하였다. 만일, 인증key가 만료되었거나 치명적인 통신 오류가 발생한다면 유지하던 인증key를 파기하고 첫 로그인 화면으로 돌아가게끔 핸들링하였다.

#### 결제 시 문자열 QR 코드 변환

사용자가 APP을 통해 결제 기능을 이용하기 위해서는 서버로부터 받아온 문자열을 QR 코드로 변환하여 화면에 출력해 주어야 한다. 이를 위해 서버로부터 받아온 문자열을 명세에 맞는 형식으로 파싱을 거친 후에, zxing 라이브러리를 활용하여 파싱이 완료된 문자열을 QR로 변환한 후 ImageView로 출력한다. 이때, 같이 받아온 만료 시간 정보를 활용하여 결제 화면에 남은 결제 시간도 같이 실시간으로 출력한다.

## K8s 클러스터

### 클러스터 개요도

텍스트, 도표, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 14 클러스터 구성 개요도

### Spring Cloud NetFlix Eureka

Eureka는 NetFlix에서 개발한 프레임워크로 현재는 Spring 재단에 기부하여 사용되고 있다. Eureka는 Service Discovery, LoadBalancing의 역할을 수행한다.

MSA에서는 서비스가 오토 스케일링이나 노드 재배치 등의 이유로 동적으로 생성되어 IP가 자주 변경되는데 이 변경되는 IP 위치와 포트를 관리하는 것이 Service Discovery이다. K8s에는 내장된 Service Discovery로 Kubernetes Dns가 있으나 새로 생성되는 서비스에 대한 탐색이 즉각적으로 반영되지 않는 문제가 존재하여 Eureka를 선택하게 되었다. 그림 15는 Eureka 상에서 관리되는 서비스와 그 인스턴스 목록이다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

그림 15 Eureka 관리 목록

### Prometheus

Prometheus는 오픈소스 시스템 모니터링 도구이다. Prometheus는 시스템 측정 정보인 메트릭을 키-값 쌍으로 타임스탬프와 함께 저장한다. TSDB(시계열 데이터베이스)에 정보를 저장하며 PromQL이라는 쿼리 언어를 통해 데이터를 조회, 시각화할 수 있다.

Prometheus는 k8s 환경에서의 표준적인 모니터링 도구로 자리 잡고 있다. 우리 프로젝트에서는 파드별 CPU 사용량을 감시하고 적절한 오토 스케일링 임곗값을 찾아서 설정하는 데 사용하였다. 또한 서비스별 파드 수를 모니터링하여 부하에 따른 파드 수의 변화를 감시할 수 있도록 구성했다.

### 오토 스케일링

클라우드의 대표적인 장점은 필요에 따라 서비스를 쉽게 확장하거나 축소할 수 있다는 것이다. Autoscaling은 이러한 클라우드의 장점을 활용하는 대표적인 기술로 CPU, 메모리, 네트워크 트래픽과 같은 값을 모니터링하여 리소스를 자동으로 조절하는 기술이다.

Scaling이란 인스턴스나 컴퓨팅 파워를 늘리는 것으로 Scale-Up과 Scale-Out이 있다. Scale-Up은 기존의 서버를 보다 높은 사양으로 업데이트하는 것을 의미한다. 수직 확장이라고도 한다. 다음으로 Scale-Out은 비슷한 사양의 서버를 하나 더 추가하는 것을 의미한다. 수평 확장이라고도 한다.

우리 팀은 서버에 Scale-Out 방식을 적용하였다. 이를 위해 서버 내부에 데이터를 저장하지 않는 무상태로 설계하였고 저장이 필요한 값은 외부 DB나 환경변수를 이용하여 파드 생성 시 주입해 주었다.

Autoscaling 임곗값을 지정할 때는 애플리케이션의 리소스 사용 추이를 잘 파악해야 한다. 우리 팀이 서버를 구현할 때 사용한 스프링 부트는 처음 실행 시 많은 CPU와 메모리를 소모하며, 이후 실행 중일 때에는 메모리는 크게 변화하지 않지만, CPU의 사용량은 급격히 줄어든다. 그리고 부하 테스트를 진행하며 알게 된 점인데 부하가 늘어날수록 큰 변화를 보인 것은 메모리가 아닌 CPU였다. 그래서 우리 팀은 Autoscaling의 기준 리소스를 CPU로 하고 서버 종류별로 임겟값을 설정하였다. 또한 처음 실행 시 CPU 소모량이 많아 임계 값을 넘어가는데 이때 Scaling이 발생하지 않도록 Deployment와 서버 각각에 Liveness와 Readiness 설정을 해주었다. 이를 설정하면 k8s는 서버 애플리케이션이 완벽하게 실행되기 전까지 파드를 정상 동작 상태로 파악하지 않아 Autoscaling과 로드밸런싱이 동작하지 않는다.

### HPA

HPA(Horizontal Pod Autoscaling)은 deployment를 자동으로 업데이트하여 Pod의 개수를 수요에 맞게 자동으로 scaling-out 해주는 Kubernetes 리소스이다. 코드 1은 사용자 서비스의 HPA 설정 파일의 일부이다.

**apiVersion**: autoscaling/v2

**kind**: HorizontalPodAutoscaler

**metadata**:

**name**: user-hpa

**…**

**spec**:

**…**

**minReplicas**: 1

**maxReplicas**: 4

**metrics**:

- **type**: Resource

**resource**:

**name**: cpu

**target**:

**type**: Utilization

**averageUtilization**: 1000

**behavior**:

**scaleDown**:

**stabilizationWindowSeconds**: 300

**policies**:

- **type**: Pods

**value**: 1

**periodSeconds**: 30

코드 1 사용자 서비스 HPA 설정파일

minReplicas와 maxReplicas는 최소 파드 수와 최대 파드 수를 의미한다. 오토 스케일링이 적용된 모든 파드가 최대로 늘어났을 때의 사용 리소스를 미리 계산하여 적절한 수치로 지정해 주어야 한다.

metric은 파드의 scale-out 지점을 설정해주는 부분이다. k8s는 수식 1에 따라서 파드 수를 조절한다. 사용자 서비스의 HPA는 시행착오를 통해 확인한 결과 처음 할당된 CPU에서 약 10배까지 사용될 때 scale-out이 되도록 설정하는 것이 적절하였다.

원하는 파드 수 = ceil[현재 파드 수 \* (현재 메트릭 값 / 원하는 메트릭 값)]

수식 1 HPA의 파드 수 계산 식

behavior에는 다양한 것들을 지정할 수 있는데 자동으로 scale-in을 지원하지 않아서 이곳에 직접 구성했다. 300초 동안 scale-out이 발생하지 않는다면 30초마다 1개의 파드를 줄인다.

# 과제 결과 및 결론

## K6와 InfluxDB를 이용한 부하 테스트

부하 테스트란 미리 결정된 부하 상태에서 시스템의 성능을 평가하는 것이다. 우리 팀은 여러 부하 테스트 도구 중 Grafana에서 개발한 도구인 k6를 사용하여 부하 테스트를 진행하였다.

k6는 부하 테스트를 쉽게 수행할 수 있도록 도와주는 오픈소스 테스트 도구이다. Javascript를 이용하여 쉽게 코드를 작성할 수 있으며 시계열 데이터베이스(TSDB)인 InfluxDB에 데이터를 쉽게 저장할 수 있다.

InfluxDB는 go로 개발된 시계열 데이터베이스로 일정 주기마다 데이터를 처리하며 새롭게 저장 혹은 자동으로 삭제한다. k6로 테스트를 수행한 결과를 저장했으며 InfluxDB를 조회할 때 사용하는 Flux 쿼리를 사용하여 Grafana에 시각화하였다.

부하 테스트 시나리오를 작성할 때 빈번하게 호출될 것으로 예상되는 API를 테스트 목표로 잡고 작성하였다. k6로 작성한 테스트의 시나리오는 아래와 같다.

1. 미리 테스트할 가상의 사용자 A와 B를 만들어 두고 A를 구매자, B를 판매자로 가정한다. 또한 B의 가게와 상품을 미리 만들어 둔다.
2. 사용자 로그인: 사용자 A와 B의 로그인 요청을 보낸다.
3. 가게 로그인: B의 가게로 로그인 요청을 보낸다.
4. 사용자 크레딧 충전: A의 크레딧을 충전하는 요청을 보낸다.
5. QR 생성: A의 구매를 위한 QR 생성 요청을 보낸다.
6. 결제: 상품의 결제 요청을 보낸다.

위의 시나리오를 통해 한 번의 반복 동안 5가지 API에 대한 총 6번의 요청이 발생한다. 테스트는 7분 동안 위 시나리오를 수행하는 virtual user 수를 30까지 늘려가며 진행하였다. virtual user는 테스트가 종료될 때까지 시나리오를 지연 없이 반복 수행한다. 테스트는 단일 파드 환경과 Autoscaling 환경을 구성하여 진행하였으며, 각각의 데이터는 virtual user 수, 5초간 평균 응답시간, TPS(Transaction per Second), 서비스별 파드 수 및 요약으로 이루어져 있다.

TPS는 1초당 처리하는 트랜잭션의 수이다. 시스템에 지연시간이 발생하지 않는다면 사용자가 지속해서 늘어나는 상황에서 TPS는 끊임없이 증가할 것이다. 그러나 시스템에는 지연시간이 존재하고 어느 순간 TPS가 증가하지 않는 순간이 발생한다. 이 지점을 Saturation Point라고 한다. 즉, Saturation Point가 높은 서비스일수록 서비스의 성능이 좋다고 판단할 수 있다. 우리가 TPS를 계산한 식은 수식 2과 같다.

TPS = (5초동안 발생한 요청 수) / 5

수식 2 TPS 계산 식

그림 16~25까지 k6를 이용해 테스트한 것을 Grafana를 이용해 시각화하였다.

* **단일 파드**

라인, 그래프, 텍스트, 도표이(가) 표시된 사진

자동 생성된 설명

그림 16 단일 파드에서 시간대별 virtual user 수

라인, 그래프, 스크린샷, 도표이(가) 표시된 사진

자동 생성된 설명

그림 17 단일 파드에서 5초간 평균 응답시간

라인, 그래프, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 18 단일 파드에서 TPS

텍스트, 라인, 그래프, 다채로움이(가) 표시된 사진

자동 생성된 설명

그림 19 단일 파드에서 전체 응답시간 산점도(API 종류별)

텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

그림 20 단일 파드에서 파드의 변화

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| API 종류 | 요청 수 | 성공률(%) | 최대  응답시간(ms) | 평균  응답시간(ms) |
| 사용자 로그인 | 2,350 | 64.0 | 3,083 | 644 |
| 가게 로그인 | 1,175 | 57.9 | 3,112 | 1,070 |
| 크레딧 충전 | 1,166 | 56.6 | 3,068 | 1,038 |
| QR 생성 | 1,166 | 56.9 | 3,062 | 939 |
| 결제 | 1,159 | 56.8 | 2,363 | 1,252 |
| 총 | 7,016 | 59.4 | 3,112 | 930 |

표 6 단일 파드 테스트 API 종류별 요약

* **오토 스케일링 적용**

라인, 그래프, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 21 Autoscaling 환경 virtual user 수

라인, 그래프, 스크린샷, 도표이(가) 표시된 사진

자동 생성된 설명

그림 22 Autoscaling 환경 5초간 평균 응답시간

라인, 그래프이(가) 표시된 사진

자동 생성된 설명

그림 23 Autoscaling 환경 TPS

텍스트, 라인, 그래프, 다채로움이(가) 표시된 사진

자동 생성된 설명

그림 24 Autoscaling 환경 전체 응답시간 산점도(API 종류별)

텍스트, 도표, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

그림 25 Autoscaling 환경 파드 변화

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| API 종류 | 요청 수 | 성공률(%) | 최대  응답시간(ms) | 평균  응답시간(ms) |
| 사용자 로그인 | 2,480 | 100 | 2,772 | 323 |
| 가게 로그인 | 1,240 | 98.7 | 3,077 | 939 |
| 크레딧 충전 | 1,240 | 98.2 | 3,091 | 1,080 |
| QR 생성 | 1,240 | 99.5 | 3,050 | 479 |
| 결제 | 1,240 | 80.2 | 3,115 | 2,021 |
| 총 | 7,440 | 96.1 | 3,115 | 861 |

표 7 Autoscaling 환경 API 종류별 요약

두 상황에서 진행된 테스트를 비교해 보았을 때 평균 응답시간은 생각보다 큰 차이를 보이지 않았다. 이는 서킷 브레이커가 3초 이상 걸리는 요청에 대해서는 모두 실패처리를 했기 때문이다. 그래서 단일 파드 환경은 50% 대의 성공률을 가지고 있다. 심지어 사용자 서비스에 너무 많은 실패한 트랜잭션이 몰려서 서비스의 네트워크가 끊기는 현상(단일 파드 환경에서 09:26:00 부분)까지 발생하였다.

TPS 또한 큰 차이를 보인다. Saturation Point를 비교해 보았을 때, 단일 파드 환경은 TPS 12에 형성된 반면 Autoscaling 환경에서는 20 초반까지 형성된 것을 볼 수 있다.

결론적으로 Autoscaling이 적용된 환경이 파드가 생성되는 데 더 많은 리소스를 사용하고 시간이 걸림에도 불구하고 더 좋은 성능을 내는 것을 알 수 있다.

## Zipkin을 이용한 트랜잭션 흐름 모니터링

MSA에서 외부에서 들어온 요청은 시스템 내부의 여러 가지 서비스를 이용한다. Zipkin은 이 요청에 특별한 헤더 값을 부여해서 요청의 전체 흐름을 파악할 수 있게 해준다. 이를 통해 시스템이 설계대로 구성되었는지 확인할 수 있고, 여러 서비스 중 개선이 필요한 서비스를 쉽게 파악할 수 있다. 그림 24는 가게 로그인 요청에 대한 흐름과 서비스별 소요 시간이 표현된 것이고, 그림 25는 Zipkin에서 데이터를 바탕으로 구성한 트랜잭션의 흐름도이다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

그림 26 Zipkin을 이용해 가게 로그인 트랜잭션을 추적한 모습

스크린샷, 블랙, 어둠이(가) 표시된 사진

자동 생성된 설명

그림 27 Zipkin이 표현한 트랜잭션 흐름도

# 개선사항

* 운영 클러스터의 분리

현재 우리의 클러스터에는 모든 리소스가 함께 올라가 동작하고 있다. 그러나 실제 실무 환경에서는 배포나 모니터링에 사용되는 리소스가 운영 중인 서비스에 영향을 미칠 수 있기 때문에 이를 분리해서 사용한다.

* 외부 저장소를 내부 저장소로 변경

본 프로젝트에서는 git repository와 image registry가 github과 docker hub로 외부에 존재한다. 외부에 대한 의존성을 낮추기 위해 이 저장소 또한 git server나 Harbor와 같은 private image registry의 형태로 클러스터 상에서 운영되어야 한다.

# 구성원별 역할 및 개발 일정

| 6월 | | | | | 7월 | | | | | 8월 | | | | | 9월 | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| 기자재 구입 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | API  endpoint 명세 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 개발 환경 구축 및  개별 서비스 구현 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | 서비스 간 통신  구현 | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | 중간보고서 작성 | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | 서비스 통합 | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 배포 및 테스트 | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  | 시스템  최적화 | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | 보약 취약점 파악 및 보완 | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 최종 보고서 작성 및 발표,심사 준비 | | | | |

표 8 개발 일정

|  |  |
| --- | --- |
| 이름 | 역할 |
| 송재원 | * CI & CD 환경 구축 및 클라우드 환경 관리 * K8s클러스터 구축 * 시스템 최적화와 테스트 |
| 김민승 | * 안드로이드 어플리케이션 개발 * 키오스크 클라이언트 개발 * 크레딧 충전 웹페이지 구현 |
| 김민종 | * 마이크로 서비스 환경 구축 및 개별 서비스 구현 * 서비스 단위 테스트와 검증 * 서비스 디스커버리, 게이트웨이 개발 |
| 공통 | * 시스템 보안성 추가 * 데이터베이스 설계 및 관리 * 기타 문서화 작업 |

표 9 구성원별 역할

# 참고 문헌

|  |
| --- |
| * [1] Spring, “Spring Cloud NetFlix”, [Online]. Available: <https://cloud.spring.io/spring-cloud-netflix/reference/html> * [2] Kubernetes, “Kubernetes Documentation”, [Online]. Available: <https://kubernetes.io/docs/home> * [3] Jenkins, “Jenkins User Documentation”, [Online]. Available: <https://www.jenkins.io/doc/> * [4] argo, “Argo CD - Declarative GitOps CD for Kubernetes”, [Online].  Available: <https://argo-cd.readthedocs.io/en/latest/> * [5] K6, “k6 documentation”, [Online]. Available: <https://k6.io/docs/> |