

|  |  |
| --- | --- |
| 분과명 | C. 네트워크/시스템 |
| 지도교수명 | 김원석 |
| 팀원 | 201723148 송재원 |
| 201606151 김민승 |
| 201724425 김민종 |

2023 전기 졸업과제 중간보고서

클라우드 기반 크레딧 거래 플랫폼

클라우드 나인

2023

내용

[1. 요구조건 및 제약사항 분석에 대한 수정사항 2](#_Toc141840036)

[1-1. 기존의 요구조건 2](#_Toc141840037)

[1-2. 변경된 요구조건 및 제약사항 2](#_Toc141840038)

[쿠버네티스 클러스터 노드 메모리 증가 2](#_Toc141840039)

[리소스 모니터링의 필요성 2](#_Toc141840040)

[GitOps적용 3](#_Toc141840041)

[2. 설계 상세화 및 변경 내역 4](#_Toc141840042)

[2-1. 배포 파이프라인 변경 4](#_Toc141840043)

[GitOps 방식의 파이프라인 설계 4](#_Toc141840044)

[Jenkins Agent의 사용 5](#_Toc141840045)

[Jib 도입 5](#_Toc141840046)

[2-2. 인증 서비스의 설계 변경 6](#_Toc141840047)

[3. 갱신된 과제 추진 계획 8](#_Toc141840048)

[4. 구성원별 진척도 8](#_Toc141840049)

[5. 보고 시점까지의 과제 수행 내용 및 중간 결과 9](#_Toc141840050)

[5-1. API 구현 및 서비스 배포 현황 9](#_Toc141840051)

[인증 서비스 10](#_Toc141840052)

[크레딧 서비스 13](#_Toc141840053)

[사용자 서비스 15](#_Toc141840054)

[가게 서비스 17](#_Toc141840055)

[5-2. 앱 디자인 및 개발 내용 20](#_Toc141840056)

# 1. 요구조건 및 제약사항 분석에 대한 수정사항

## 1-1. 기존의 요구조건

본 과제의 목표는 ‘클라우드 환경의 크레딧 거래 플랫폼’을 마이크로서비스 아키텍처를 적용하여 구현하는 것이다. 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 마이크로서비스 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.

## 1-2. 변경된 요구조건 및 제약사항

### 쿠버네티스 클러스터 노드 메모리 증가

초기에 개발환경 쿠버네티스 클러스터를 상용 클라우드 환경에 구축하는 것은 비용 문제 및 보안 문제가 발생할 수 있어서 온프레미스 환경에 클러스터를 구축하고 개발을 진행했다. 2GB의 메모리를 사용하는 3개의 가상머신을 생성하고, Kubeadm이라는 온프레미스 환경 쿠버네티스 클러스터 구성 도구를 사용하여 클러스터를 구축하였다. 그리고 서비스에 필요한 리소스 및 애플리케이션을 배포하였는데, 테스트 운영 중 메모리 부족 문제가 발생하여 Pod가 종료되는 현상이 지속해서 발생했다.

실제 서비스 환경이 구성될 AWS EC2의 t3.small 인스턴스 또한 2GB의 메모리를 사용한다. 이 환경에 그대로 배포할시 동일한 메모리 부족 문제가 발생할 것으로 예상되어 개발환경의 가상머신 메모리를 4GB로 증가시키고, 서비스 환경의 인스턴스 또한 t3.medium으로 변경하여 4GB의 메모리로 구성해야 한다.

### 리소스 모니터링의 필요성

요구조건 중 서비스는 자유로운 scale-out이 가능하도록 설계되어 있다. 그러나 클러스터 상에서 자동으로 scale-out을 적용하는 autoscaling을 적용하기 위해서는 서비스의 사용 리소스 양에 임계 값을 지정해주어야 한다. 이에 따라서 서비스가 기본적으로 사용하는 리소스의 양을 모니터링하여 부하가 많이 걸리는 요소에 대해 임계 값을 지정하여 임계 값을 넘을 시 자동으로 scale-out하도록 설정해주어야 한다. 그러므로 autoscaling을 적용하기 위해서는 서비스별 리소스 사용 모니터링이 필요하다.

### GitOps적용

프로젝트가 지향하는 클라우드 네이티브 환경은 유연하며 언제든 확장, 변경할 수 있어야 한다. 이에 따라 쿠버네티스 클러스터를 구성하는 리소스는 자주 생성, 수정, 삭제되며, 쉽게 관리될 수 있어야 한다. 현재 구성된 클러스터의 리소스는 CLI를 통해 생성된 것으로 유지보수가 어려워 추구하는 바와 맞지 않는다. 그러므로 DevOps 방법론 중 하나인 GitOps를 프로젝트에 적용하고자 한다.

GitOps란 Git repository의 코드와 배포 환경이 일치하도록 자동화된 프로세스를 구축하는 것을 의미한다. 기존의 프로그램을 업데이트할 때 Git repository의 설정 파일만 수정하여도 자동화된 프로세스를 통해 일관된 환경을 유지할 수 있다.

GitOps를 적용한다면 다음과 같은 장점이 있다.

1. 배포 결과를 즉시 확인할 수 있으며, Git 형상 관리를 통해 빠르게 배포할 수 있다.

2. 배포가 실패하거나 에러가 발생했을 때 Git의 버전 관리를 이용해서 빠르게 안정적인 버전으로 복구할 수 있다.

3. 쿠버네티스 클러스터에 직접 접근하지 않고 git repository와 image registry만 관리하면 돼서 보안 측면에서 보다 안전하다.

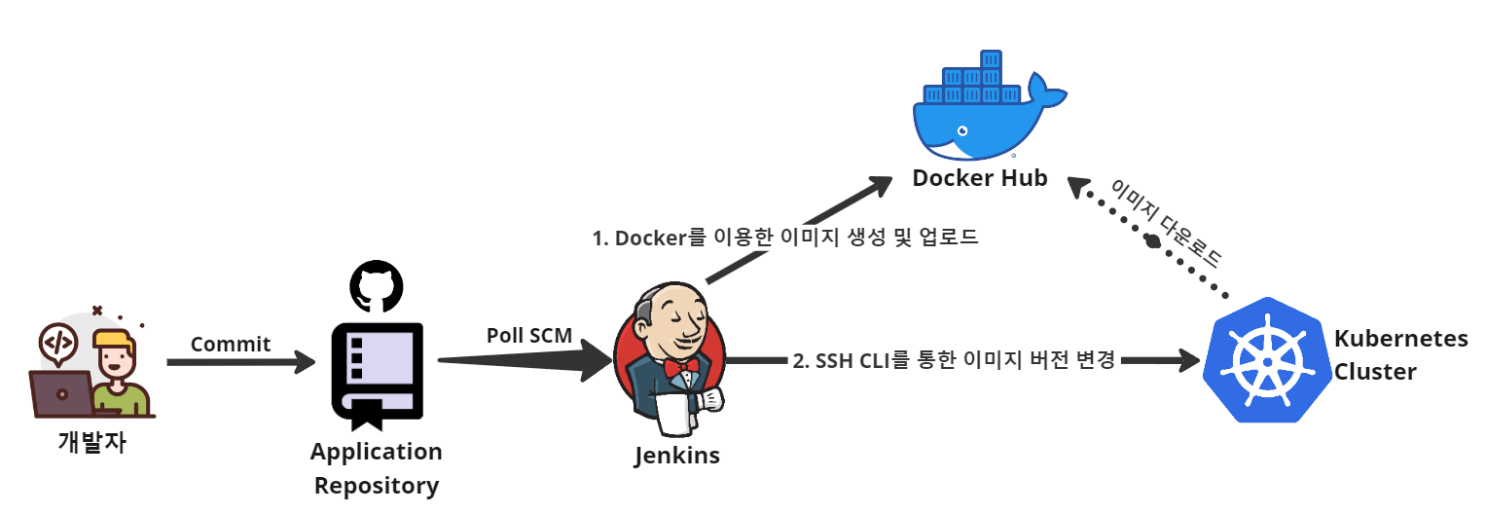
4. Commit 메시지나 pull request를 통해 변경 이력을 쉽게 남기고 기록할 수 있다.

5. 모든 것이 기록된 Git repository 통해 쿠버네티스 클러스터 인프라의 변화와 구성을 팀 내에서 쉽게 학습 및 공유할 수 있다.

# 2. 설계 상세화 및 변경 내역

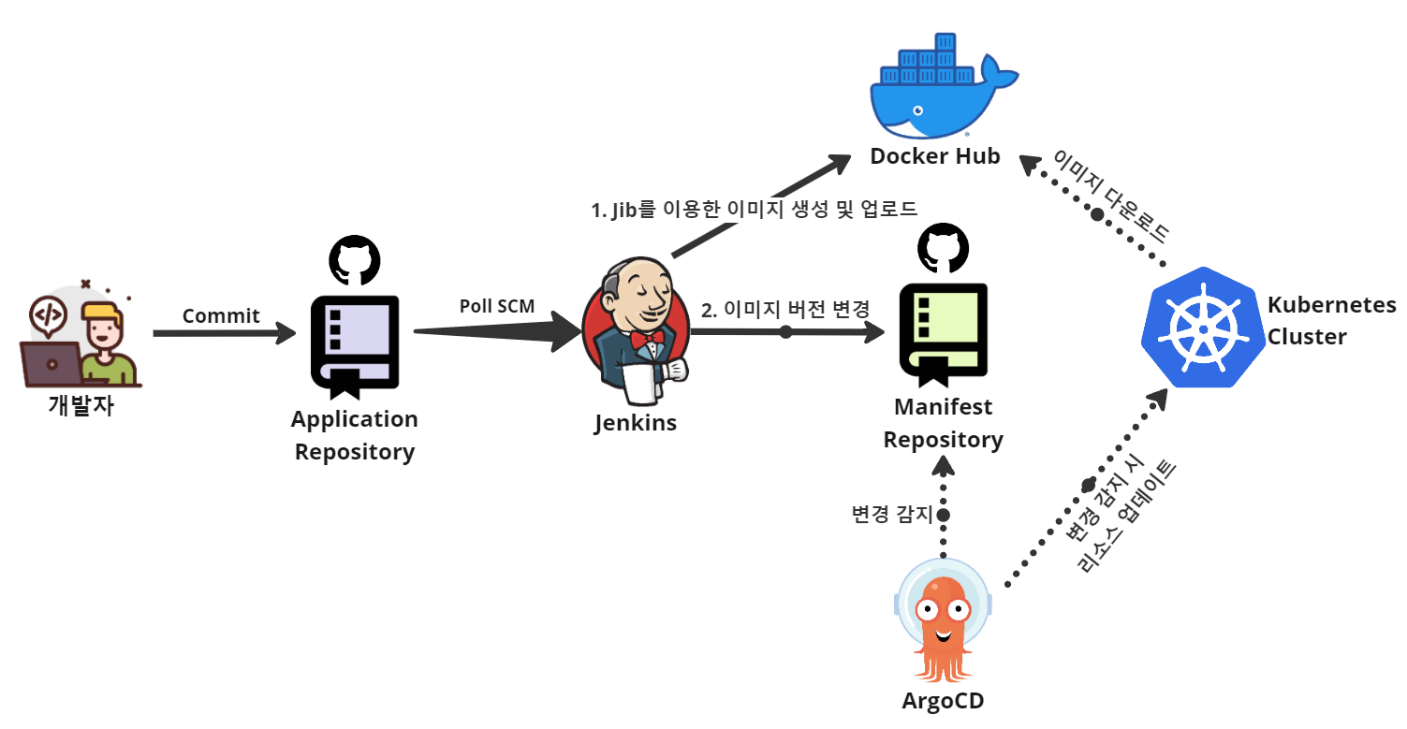
## 2-1. 배포 파이프라인 변경

### GitOps 방식의 파이프라인 설계



[그림 1] 이전의 배포 파이프라인

기존의 설계에서는 Jenkins에서 SSH CLI를 통해 클러스터에 직접적으로 배포하였다. 이 방식을 사용 시 Jenkins에 클러스터의 접근 권한을 줘야 했고, Jenkins에서 너무 많은 일을 처리해야 해서 배포 시간이 오래 걸렸다. 또한 새로운 배포에서 오류가 발견된다면 수동으로 복구해야 했다.



[그림 2] 변경된 배포 파이프라인

새로운 배포 파이프라인을 보면 클러스터의 리소스를 변경하는 책임을 GitOps 도구인 ArgoCD로 넘겼다. Jenkins는 Github repository와 Image Hub만 관리하면 된다.

### Jenkins Agent의 사용

### Jenkins는 지속적인 통합(CI)과 지속적인 배포(CD)를 지원하는 도구이다. Jenkins는 Master Node와 Agent Node로 구성되어 있으며, Master node가 Jenkins Pipeline으로 정의된 모든 흐름을 관장하는 컨트롤 서버의 역할을 담당하고 Jenkins Agent에 일을 분배한다. Jenkins Agent는 개별적인 서버로 존재하는 Agent Nodes와 작업이 필요할 때 컨테이너의 형태로 생성되어 Dynamic Agent라고도 불리는 Agent Clouds로 구분된다.

### 기존의 설계에서는 Agent를 사용하지 않고 개별적인 서버에 존재하는 Master Node에서 모든 배포 작업을 수행하는 형태였다. 그러나 이 경우에는 작업이 서로 겹치며 충돌이 발생할 수도 있고, 병렬적인 작업의 수행이 불가능하다는 단점이 존재한다.

문제를 해결하기 위해 쿠버네티스 클러스터 내에 Jenkins를 구성하고, 작업이 발생했을 때 동적으로 Jenkins Agent Pod를 생성하여 Job을 수행하는 형태로 변경하였다. 이를 통해서 동시에 MSA의 개별 Service를 빌드 및 배포할 수 있게 되었고, 서버 수를 줄여 리소스 비용을 절약했다.

### Jib 도입

MSA를 구현할 때 컨테이너 기술은 필수 불가결하다. 컨테이너 기술을 지원하는 플랫폼 중 가장 유명한 것은 Docker로 컨테이너의 생성 및 관리, 배포까지 지원하는 플랫폼이다. 그러나 이미지 생성을 위해서는 Dockerfile을 작성해야 하고 이미지 빌드 시에는 Docker가 설치되어 있어야 한다.

프로젝트의 초기 설계에서도 Jenkins 컨테이너에 DinD(Docker-in-Docker)를 적용하여 Jenkins에서 직접 Docker 데몬을 사용해 이미지를 빌드하고 배포하였다. 그러나 Jenkins에 부여된 리소스가 크지 않아 DinD로 인해 성능저하가 크게 발생했다. 게다가 Jenkins에 Dynamic Agent를 적용한 뒤에도 Agent Pod가 생성될 때마다 Docker를 설치해야 했다.

이에 구글에서 제공하는 오픈소스 컨테이너 이미지 빌더인 Jib을 도입하기로 하였다. Jib은 Maven과 Gradle 플러그인으로 제공된다. 간편하게 이미지를 생성, 배포할 수 있으며, 변경된 부분만 새로 빌드해서 배포 시간이 단축된다. 그리고 OCI(Open Container Initiative) 표준에 호환되는 이미지를 생성하여 Docker뿐만 아니라 다양한 컨테이너 런타임에서도 실행할 수 있다.

## 2-2. 인증 서비스의 설계 변경

텍스트, 도표, 만화 영화이(가) 표시된 사진

자동 생성된 설명

[그림 3] 기존 인증 서비스 설계

[그림 3]은 기존 설계에서 사용자 결제 발생 시 가게 서비스가 사용자의 개체식별번호를 획득하는 순서도이다. 그림을 보면 인증 서비스는 QR 코드 정보의 생성 및 인증을 모두 담당하고 있다. 이 설계에서는 QR 코드를 생성하거나 인식할 때마다 인증 서비스에 트래픽이 발생해서 병목 현상이 일어날 것이다. 이에 기존의 설계를 변경하여 인증 서비스를 CA(Certificate Authority)로 하는 인증서 기반 키 교환 방식을 적용하였다.

텍스트, 도표, 만화 영화이(가) 표시된 사진

자동 생성된 설명

[그림 4] 변경된 인증 서비스 설계

|  |  |  |
| --- | --- | --- |
| 가게 서비스 생성 시 발생 | | |
| 0 | 인증 서비스 공개키 제공 | PUa → 가게 서비스 |
| 사용자 서비스 생성 시 발생 | | |
| 1 | 공개키, 서비스 정보 전달 | { PUu, ServiceInfo } → 인증 서비스 |
| 2 | 인증 서비스 비밀키로 인증서 생성 | EN(PRa, { PUu, ServiceInfo }) |
| 3 | 인증서 발급 | EN(PRa, { PUu, ServiceInfo }) → 사용자 서비스 |
| QR 생성 요청 시 발생 | | |
| 4 | QR 생성 요청 발생 | - |
| 5 | 사용자 서비스 비밀키로 식별번호 암호화 | EN(PRu, SNu) |
| 6 | 인증서와 암호화된 식별번호 전달 | { EN(PRa, { PUu, ServiceInfo }), EN(PRu, SNu) } → 클라이언트 |
| 7 | json 데이터로 QR 생성 | - |
| QR 인식 시 발생 | | |
| 8, 9 | 가게 서비스에서 QR 인식 | { EN(PRa, { PUu, ServiceInfo }), EN(PRu, SNu) } → 가게 서비스 |
| 10 | 인증 서비스 공개키로 인증서 복호화 | DE(PUa, EN(PRa, { PUu, ServiceInfo })) -> PUu 획득 |
| 11 | 사용자 서비스 공개키로 식별번호 복호화 | DE(PUu, EN(PRu, SNu)) -> SNu 획득 |

[표 1] 인증 서비스 설계 설명

[표 1]은 변경된 설계에 대한 설명을 제공한다. 설명 옆에는 기호로 표기하였는데, EN(키, 데이터)는 키를 이용해 데이터를 암호화하는 것을 의미하고 DE(키, 데이터)는 키를 통해 데이터를 복호화하는 것을 의미한다. 그리고 PRa, PUa는 인증 서비스의 비밀키와 공개키를, PRu, PUu는 사용자 서비스의 비밀키와 공개키를, SNu는 사용자 식별번호를 나타낸다.

이 설계의 변경으로 얻을 수 있는 이점은 인증 서비스와 개별 서비스 간 통신이 서비스 생성 시에만 발생한다는 것이다. 개별 서비스의 생성 시, 공개키, 인증서를 획득하는 작업이 완료되면 그 이후에는 개별 서비스에서 인증서를 이용하여 데이터를 주고받을 수 있다.

# 3. 갱신된 과제 추진 계획

| 6월 | | | | | 7월 | | | | | | 8월 | | | | | | 9월 | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1주 | 2주 | 3주 | 4주 | 5주 | 1주 | 2주 | 3주 | 4주 | 5주 | 1주 | | 2주 | 3주 | 4주 | 5주 | 1주 | | 2주 | 3주 | 4주 | 5주 | |
| 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 2] 과제 추진 계획

# 4. 구성원별 진척도

|  |  |
| --- | --- |
| 이름 | 역할 |
| 송재원 | * 온프레미스 환경 배포 파이프라인 구축 |
| 김민승 | * 사용자 앱 디자인 및 구현 |
| 김민종 | * SpringBoot를 이용한 RESTful API 구현 |

[표 3] 구성원별 진척도

# 5. 보고 시점까지의 과제 수행 내용 및 중간 결과

## 5-1. API 구현 및 서비스 배포 현황

|  |  |  |  |
| --- | --- | --- | --- |
|  | 이름 | Endpoint | 응답 페이로드 |
| 사용자 서비스 | 사용자 정보 조회 | GET /user-service/api/user | 아이디, 이름, 이메일, 크레딧 잔액 |
| 사용자 정보 수정 | PUT /user-service/api/user | 아이디, 이름, 이메일 |
| 사용자 탈퇴 | DELETE /user-service/api/user | - |
| 회원가입 | POST /user-service/api/user/signup | - |
| 아이디 검증 | GET /user-service/api/user/check-duplicate | - |
| 로그인 | POST /user-service/api/login | 사용자 JWT 토큰 |
| QR 정보 생성 | GET /user-service/api/qr | 인증서, 암호화된 사용자 개체식별번호 |
| 크레딧 충전 | POST /user-service/api/credit | - |
| 가게  서비스 | 가게 등록 | POST /store-service/api/store | 가게 목록 |
| 가게 조회 | GET /store-service/api/store | 가게 목록 |
| 가게 로그인 | POST /store-service/api/{가게ID}/login | 가게 JWT 토큰 |
| 상품 추가 | POST /store-service/api/{가게ID}/product | 상품 목록 |
| 상품 조회 | GET /store-service/api/{가게ID}/product | 상품 목록 |
| 상품 업데이트 | POST /store-service/api/{가게ID}/product/{상품ID} | 수정된 상품 정보 |
| 상품 삭제 | DELETE /store-service/api/{가게ID}/product/{상품ID} | 상품 목록 |
| 상품 판매 | POST /store-service/api/{가게ID}/product/sale | 메뉴 선택 정보 |

[표 4] API 구현 목록

위 표는 현재 구현한 API 목록이다. 내부 로직은 완벽하게 구현되어 있지 않으나, 임의의 값을 응답하는 형태로 API를 제공한다. Endpoint는 서비스 간 통신에 필요한 endpoint와 외부에 노출된 클라이언트에서 사용하는 endpoint로 구분하였다. 그리고 쿠버네티스 클러스터의 L7 Gateway 역할을 수행하는 Ingress 리소스를 구성하여 외부에서는 서비스 간 통신에 사용되는 endpoint에 접근하지 못하도록 만들었다.

모든 서비스는 개별적인 Github repository에 저장되며 main branch에 merge가 발생할 때마다 Jenkins에서 빌드가 일어난다. Jenkins에서 새로운 이미지를 도커 레지스트리에 업로드한 후, ArgoCD가 감시하고 있는 Manifest repository의 이미지 버전을 변경한다. 이에 따라 ArgoCD는 변경된 리소스를 쿠버네티스 클러스터에 반영한다.

네 개의 서비스 모두 기본적으로 하나의 Service, Deployment, Pod 리소스를 가지고 있다. Pod는 실질적으로 동작하는 컨테이너를 의미한다. Pod는 지속적으로 제거 및 삭제되는데, 이때 네트워크 정보도 계속해서 바뀌게 된다. Service 리소스는 Pod의 네트워크 정보가 변경되어도 정상 통신이 가능하도록 네트워크 추상화를 제공한다. 또한 트래픽을 골고루 분산해주는 로드 밸런싱도 수행한다. Deployment 리소스는 Pod를 배포하고 관리하는 역할을 담당한다. 배포 시 서비스 다운타임을 최소화하는 다양한 배포 전략을 지원하며, Pod가 예상치 못한 에러로 다운되었을 때, 복구하는 역할도 수행한다.

이제 서비스별로 자세한 배포 상태와 주요 API를 설명하겠다.

### 인증 서비스

텍스트, 폰트, 도표, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[그림 5] ArgoCD로 본 인증 서비스 배포 상태

[그림 5]는 ArgoCD로 확인한 인증 서비스 배포 상태이다. ‘auth’라는 이름의 Service 리소스와 ‘auth-deployment’라는 이름의 Deployment 리소스가 배포되어 있고, 2개의 Pod가 존재하는 것을 볼 수 있다. 이유는 다른 서비스 생성 시 필연적으로 인증 서비스에 트래픽이 발생하는데, 개발 환경에서 서비스의 생성은 빈번하게 발생한다. 이에 개발환경에서 2개의 Pod를 사용했다.

다음으로 인증 서비스의 주요 API를 소개한다.

#### 식별번호 생성

텍스트, 번호, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

[그림 6] 식별번호 생성 API 요청 및 결과

식별번호는 모든 서비스를 통틀어 단 하나만 존재하는 유일한 값이다. 이 값을 생성하고 관리하는 역할을 인증 서비스에서 담당한다.

#### 인증 서비스 공개키 획득

텍스트, 스크린샷, 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

[그림 7] 인증 서비스 공개키 획득 API 요청 및 결과

인증 서비스의 공개키를 획득할 수 있는 API이다. 다른 서비스에서 인증서를 복호화할 때 사용한다.

#### 인증서 생성

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

[그림 8] 인증서 생성 API 요청 및 결과

다른 서비스가 생성될 때 만들어질 때 사용된다. 다른 서비스의 정보와 공개키를 받아서 인증 서비스의 비밀키로 암호화한다.

### 크레딧 서비스

텍스트, 폰트, 도표, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[그림 9] ArgoCD로 본 크레딧 서비스 배포 상태

‘credit’이라는 이름의 Service 리소스와 ‘credit-deployment’라는 이름의 Deployment 리소스가 배포되어 있고, 1개의 Pod가 존재하는 것을 볼 수 있다.

다음으로 크레딧 서비스의 주요 API이다.

#### 계좌 생성

텍스트, 소프트웨어, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

[그림 10] 계좌 생성 API 요청 및 결과

식별번호를 경로에 포함해 요청을 보내면 해당 식별번호에 대한 계좌를 생성한다.

#### 크레딧 충전

텍스트, 번호, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[그림 11] 크레딧 충전 API 요청 및 결과

식별번호를 경로에 포함하고 충전 정보를 함께 보내면 해당 식별번호의 계좌에 크레딧을 충전한다.

#### 송금

텍스트, 번호, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

[그림 12] 크레딧 송금 API 요청 및 결과

송금자와 수령자의 식별번호를 경로에 포함하고 송금 정보를 함께 보내면, 송금자의 크레딧은 감소하고 수령자의 크레딧은 증가한다.

### 사용자 서비스

스크린샷, 텍스트, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

[그림 13] ArgoCD로 본 사용자 서비스 배포 상태

‘user’라는 이름의 Service 리소스와 ‘user-deployment’라는 이름의 Deployment 리소스가 배포되어 있고, 1개의 Pod가 존재하는 것을 볼 수 있다.

다음으로 사용자 서비스의 주요 API이다.

#### 회원가입

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

[그림 14] 회원가입 API 요청 및 결과

아이디, 사용자이름, 비밀번호, 이메일을 입력받아 사용자를 생성한다.

#### 로그인

텍스트, 번호, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[그림 15] 로그인 API 요청 및 결과

아이디와 비밀번호를 입력받아서 사용자 JWT 토큰을 쿠키로 발급한다.

#### QR 정보 생성

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

[그림 16] QR 정보 생성 API 요청 및 결과

사용자 JWT 토큰을 받아서 사용자의 식별번호를 암호화하여 인증서와 함께 응답한다.

### 가게 서비스

텍스트, 도표, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

[그림 17] ArgoCD로 본 가게 서비스 배포 상태

‘store’라는 이름의 Service 리소스와 ‘store-deployment’라는 이름의 Deployment 리소스가 배포되어 있고, 1개의 Pod가 존재하는 것을 볼 수 있다.

다음으로 가게 서비스의 주요 API이다.

#### 가게 등록

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

[그림 18] 가게 등록 API 요청 및 결과

사용자 JWT 토큰을 받아서 해당 사용자 식별번호에 대한 가게를 생성한다.

#### 상품 추가

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

[그림 19] 상품 추가 API 요청 및 결과

가게 ID를 포함한 경로에 사용자 JWT 토큰과 상품 정보를 받아서 해당 가게에 상품을 추가한다.

#### 가게 로그인

텍스트, 소프트웨어, 번호, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

[그림 20] 가게 로그인 API 요청 및 결과

가게 ID가 포함된 경로에 사용자 JWT 토큰을 받아서 가게 JWT 토큰을 발급한다. 사용자 JWT 토큰은 유효기간이 짧은 반면, 가게 JWT 토큰은 유효기간이 긴 대신 동일한 IP에서만 유효하다.

#### 상품 판매

텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[그림 21] 상품 판매 API 요청

QR 인식 정보와 함께 구매를 결정한 상품에 대한 리스트를 제공한다. QR 인식 정보에는 인증서와 암호화된 식별번호가 들어있으며, 상품 정보는 상품 ID와 수량으로 이루어져 있다.

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

[그림 22] 상품 판매 API 결과

상품 판매 요청에 대한 응답으로 구매한 상품의 리스트와 함께 총 구매 금액을 응답한다.

## 5-2. 앱 디자인 및 개발 내용

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

[그림 23] 사용자 앱 시나리오

위 그림은 사용자 앱의 시나리오이다. 앱은 현재 시나리오대로 형태만 구현하였으며, 서버의 API가 완전히 개발된 후 API 요청 로직을 작성할 예정이다. 아래의 이미지는 실제 구현한 앱의 화면이다.

텍스트, 스크린샷, 디자인, 템플릿이(가) 표시된 사진

자동 생성된 설명

[그림 24] 왼쪽 위부터 로그인 페이지, 메인 페이지, 회원가입 페이지,   
크레딧 충전 페이지, 결제 완료 페이지, 결제 페이지

버튼 클릭 등 각 특정 동작마다 실행될 코드블록 틀을 구축하여 추후 타 서비스 및 API 적용 시 각 코드블록에 바로 필요한 코드를 작성해 넣어 바로 구현할 수 있도록 만들었다. 샘플 문자열을 QR코드로 변환하고 출력하는 기능을 구현하였다. 추후 서버 통신으로 받아온 문자열에 적용하여 출력한다.