캡스톤 디자인 I 최종결과 보고서

프로젝트 제목(국문): 백엔드 서비스 프로바이더: 컨테이너 오케스트레이션 플랫폼 개발
프로젝트 제목(영문): Backend Service Providers: Container Orchestration Platform Development

프로젝트 팀(원): 학번: 20211939 이름: 허유정
프로젝트 팀(원): 학번: 20217140 이름: 김창인

1. 중간보고서의 검토결과 심사위원의 '수정 및 개선 의견'과 그러한 검토의견을 반영하여 개선한 부분을 명시하시오.

가. 이미지 빌드 구현의 필요

- 자체적인 컨테이너 오케스트레이션 플랫폼 개발을 위해서는 다른 도구의 사용 없이 이미지 빌드부터 동작해야 한다.
- 따라서 보편적으로 쓰이는 컨테이너 기술을 사용하지 않고 본 프로젝트를 진행하는 Linux 환경에서의 이미지 생성이 필요하다.
- 빌드해야하는 경로에서 “Carte build”를 입력할 시, layer에 따라 이미지가 생성되는 환경을 개발하였다.

나. CI/CD의 배포방안

- 개발 후, 버전 관리를 하며 배포를 할 수 있어야 한다.
- 이에 Git을 버전 관리도구를 사용하고, GitHub 플랫폼에 저장소를 호스팅할 필요성이 있다.
- GitFlow의 브랜칭 전략을 사용하여 기능 개발, 버그 수정, 릴리즈 관리를 통한 저장소를 관리 하는 중이다.
- 메인 브랜치, 기능 브랜치가 변경될 시 자동 빌드 프로세스가 트리거되어 코드 통합과 일관성을 유지하도록 한다
- 기존 플랫폼을 사용하지 않고 고유의 빌드 시스템을 사용하여 컨테이너 이미지 생성이 가능하며 이는 오픈 소스 툴이나 자체 개발 도구 사용이 가능하다

다. 컨테이너 기술을 사용한 메타버스 게임 서버 구축

- 다수의 동접자를 수용할 수 있는지 테스트 하기 위해 공학설계입문 수업에서 이를 활용하였다.
- 애플리케이션 운영을 표준화하고, 코드를 원활하게 이동하며 리소스 사용률을 높이기 위해 컨테이너 기술을 사용하였다.
- 로그 파일을 통해 원활한 운영 과정을 확인할 수 있으며 예외 상황을 식별할 수 있다
- 컨테이너 기술 사용으로 더 많은 코드를 실행하여 리소스 사용률을 높일 수 있었으며 이는 비용 절감의 효과를 볼 수 있다

2. 기능, 성능 및 품질 요구사항을 충족하기 위해 본 개발 프로젝트에서 적용한 주요 알고리즘, 설계방법 등을 기술하시오.

가. 컨테이너 이미지 빌드 시스템

1) Carte CLI
- 명령줄 인터페이스를 구성하여 리눅스 환경에서 명령어(Carte build, Carte run)를 실행한다.

2) Cartefile 기반 이미지 생성
- 각 애플리케이션을 컨테이너화하기 위해 Cartefile 내용을 사용한다. 각 명령어 별로 레이어 계층을 나누어 소스코드 복사, 실행파일, 셋업 쉘 파일, 메타데이터 등을 포함한다.

나. 메타버스 게임 서버

1) GIN Framework
- 메타버스 게임 서버 개발에 빠르고 효율적인 라우팅 제공을 위해 GIN 프레임워크를 사용한다.

2) MongoDB
- 게임의 다양한 데이터, 특히 맵 정보와 에셋 관리를 위해 비정형 데이터 저장에 적합하고 높은 확장성을 제공하는 MongoDB를 사용한다.

3) MySQL
- 사용자 인증 및 관리를 위한 Login 서버는 관계형 데이터베이스의 데이터 무결성 및 관리 기능을 활용하기 위해 MySQL을 사용한다.

4) 엔드포인트 구성

가) Login Controller
- (POST) ip:8000/signup
- (POST) ip:8000/login

나) Map Controller
- (POST) ip:8070/map_data
- (GET) ip:8070/map_data
- (GET) ip:8070/maplist

다) Asset Controller
- (POST) ip:8080/asset_upload
- (GET) ip:8080/asset_search
- (GET) ip:8080/asset_info
- (GET) ip:8080/asset_down

3. 요구사항 정의서에 명세된 기능 및 품질 요구사항에 대하여 최종 완료된 결과를 기술하시오.

가. 전체 시스템 구성도

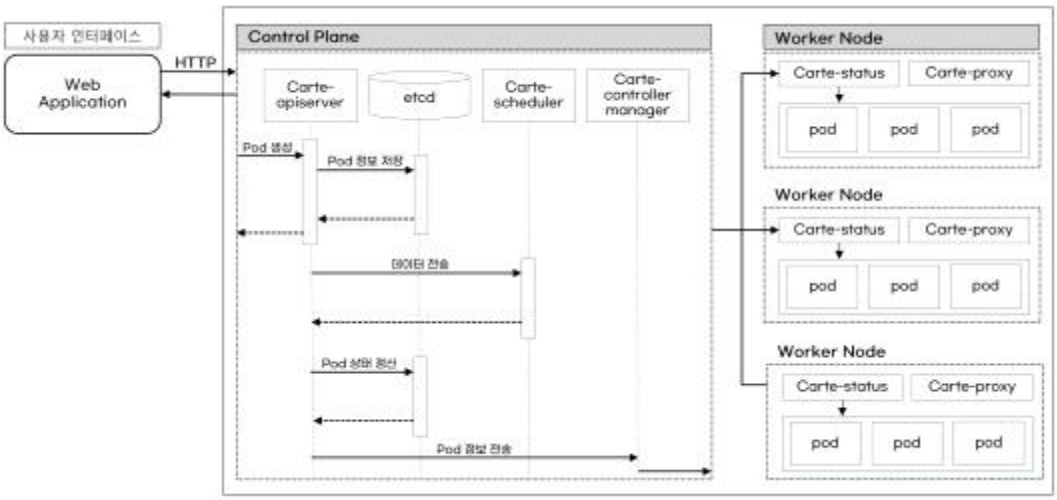


그림 1 시스템 구성도(컨테이너 오케스트레이션 플랫폼)

- 그림1은 전체적인 프로젝트 구성도를 나타낸 것이며, 컨테이너 런타임 관리, 스케일링, 네트워킹의 내용을 포함하고 있다.
- 컨테이너 라이프사이클을 쉽게 제어하고, 필요에 따라 리소스를 동적으로 조정 가능하며 컨테이너 간 외부 네트워크와의 연결을 용이하게 할 수 있다.

나. Carte Engine(Image build)

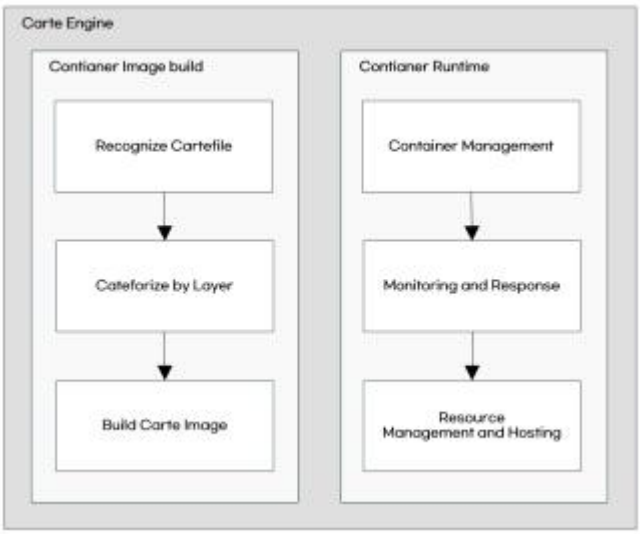


그림 2 Carte Engine(Image build)

- 그림2는 컨테이너 오케스트레이션 플랫폼 개발을 위해 자체적으로 제작한 이미지 빌드 과정을 명시한 것이다

- 컨테이너 실행을 위해서는 이미지 생성과정이 필요하며, 본 프로젝트에서 개발한 메타버스 서버에 최적화된 이미지 빌드를 구현한다.

다. 메타버스 게임 서버

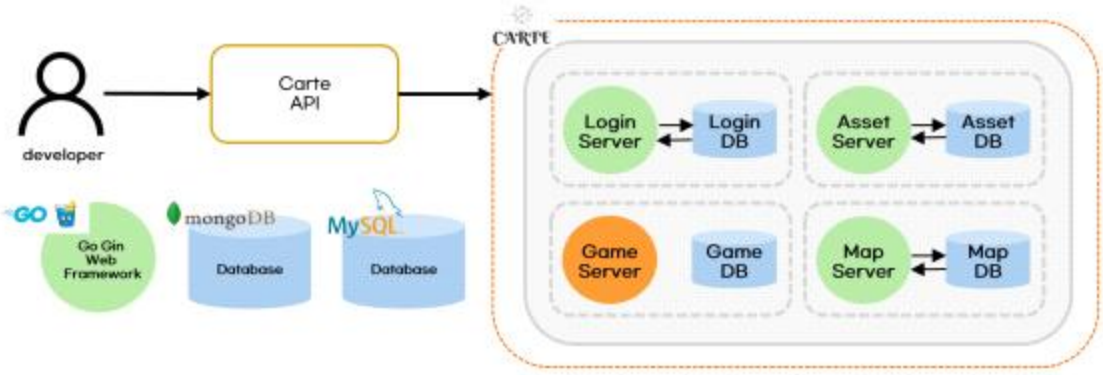


그림 3 메타버스 게임 구조도

- 그림3은 컨테이너 오케스트레이션 플랫폼의 기능 검증을 위해 실시간 멀티플레이어 게임 서버 구축 내용을 포함하고 있다.
- 이를 통해 플랫폼의 성능, 안정성, 확장성을 평가하고 최종적으로 고성능, 고가용성을 보장하는 게임 서버 인프라의 구현이 가능하다.

라. 사용자 테스트

- 공학설계입문 수업에서 약 40명의 학생을 대상으로 테스트를 진행하였다.
- 사용자 관점에서 경험 가능한 상황을 확인하고 개선 포인트를 도출할 수 있으며 다수의 사용자가 동시에 Login과 여러개의 Asset Down을 통해 사용성을 확인할 수 있었다

기능 요구사항

요구사항 고유번호		SFR-001		
요구사항 명칭		컨테이너 이미지 빌드		
요구사항 분류		기능	응락수준	필수
요구사항 상세설명	정의	애플리케이션을 컨테이너 이미지로 빌드하는 기능		
	세부내용	① 다양한 언어와 프레임워크로 작성된 애플리케이션을 컨테이너 이미지로 빌드할 수 있음 ② 애플리케이션 코드와 종속성을 포함하여 컨테이너 이미지 생성 ③ 이미지 버전 관리 및 레지스트리 저장 기능 ④ 레이어 계층을 분류하여 최종 빌드 이미지 생성 ⑤ 이미지 빌드 자동화 및 배포 파이프라인 연동		

요구사항 고유번호		SFR-002		
요구사항 명칭		컨테이너 런타임		
요구사항 분류		기능	응락수준	필수
요구사항 상세설명	정의	실행 중인 컨테이너에 대한 기본적인 관리 기능		
	세부내용	① 컨테이너의 생성, 시작, 중식, 삭제의 런타임 관리 기능 ② 컨테이너 이상 상태 감지시, 적절한 조치 취할 수 있음 ③ 다양한 컨테이너 이미지 호스팅 및 관리 기능 ④ 컨테이너 실패할 경우 자동 재시작 기능 ⑤ 컨테이너 간의 리소스 관리와 격리 제공으로 성능, 안정성 부여		

요구사항 고유번호		SFR-003		
요구사항 명칭		컨테이너 오케스트레이션(네트워킹, 스케일링)		
요구사항 분류		기능	응락수준	필수
요구사항 상세설명	정의	컨테이너 배포, 관리, 확장을 위한 오케스트레이션 기능 제공		
	세부내용	① 컨테이너 네트워킹: 컨테이너 간 통신, 부하 분산, 서비스 디스커버리 ② 컨테이너 스케일링: 자동 스케일링, 수평/수직 확장 ③ 컨테이너 배포: 롤링 업데이트		

인터페이스 요구사항

요구사항 고유번호		SIR-001		
요구사항 명칭		CLI		
요구사항 분류		인터페이스	응락수준	필수
요구사항 상세설명	정의	사용자가 명령줄에서 컨테이너 관리 기능을 사용할 수 있는 CLI 제공		
	세부내용	① 컨테이너 이미지 생성 및 확인 등의 기본 명령 지원 ② 컨테이너 생성, 시작, 중지, 삭제 등의 기본 명령 지원 ③ 컨테이너 오케스트레이션 기능 CLI 인터페이스 제공 ④ 사용자 정의 스크립트 작성 및 실행 기능 지원		

보안 요구사항

요구사항 고유번호	SER-002		
요구사항 명칭	네트워크 보안		
요구사항 분류	보안	응락수준	선택
요구사항 세부내용	① 외부와의 네트워크 통신 보안 정책 관리 기능 ② 컨테이너 간의 통신을 안전하게 보호하기 위한 네트워크 보안 프로토콜 ③ 네트워크 트래픽 암호화 지원 ④ 네트워크 보안 정책 정의 및 적용할 수 있는 기능 제공을 통한 시스템 전체적인 보안 수준 유지 ⑤ 보안 감사를 통한 네트워크 사용에 대한 모든 활동 추적 및 보안 위협 탐지		

프로젝트 관리 요구사항

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

테스트 요구사항

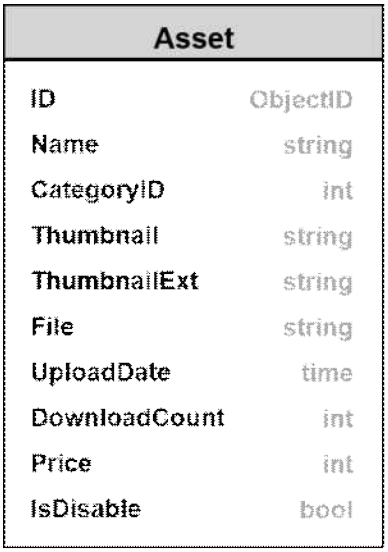
요구사항 고유번호	TER-001		
요구사항 명칭	Login Server		
요구사항 분류	테스트	응락수준	필수
요구사항 세부내용	- 사용자 접속 정보 관리 기능 - 입력된 사용자 정보 토대로 일치 여부 확인 기능		

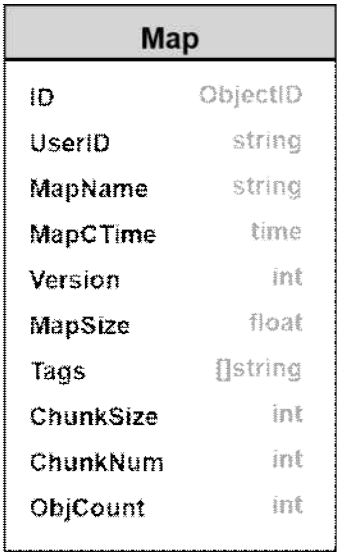
요구사항 고유번호	TER-002		
요구사항 명칭	Asset Server		
요구사항 분류	테스트	응락수준	필수
요구사항 세부내용	- asset 업로드 및 다운로드 관리 - asset 정보 관리 기능 - 카테고리별 에셋 조회 관리		

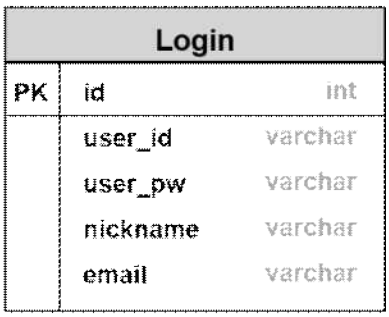
요구사항 고유번호	TER-003		
요구사항 명칭	Map Server		
요구사항 분류	테스트	응락수준	필수
요구사항 세부내용	- 다중 사용자 접속 가능 map - 제작자와 map데이터를 기준으로 정보 저장 기능		

요구사항 고유번호	TER-004		
요구사항 명칭	Game Server		
요구사항 분류	테스트	응락수준	필수
요구사항 세부내용	- 다중 사용자 로그인 확인 - 여러 사용자와의 소통 기능 및 실시간 정보 업데이트 가능		

DB 스키마(게임서버)







4. 구현하지 못한 기능 요구사항이 있다면 그 이유와 해결방안을 기술하시오,

최초 요구사항	구현 여부(미구현, 수정, 삭제 등)	이유(일정부족, 프로젝트 관리미비, 팀원변동, 기술적 문제 등)
컨테이너 이미지 빌드		해당사항 없음
컨테이너 런타임	미구현	캡스톤 디자인2에서 구현
컨테이너 오케스트레이션	미구현	캡스톤 디자인2에서 구현
CLI		해당사항 없음, 기능 추가 후 확장
네트워크 보안	미구현	캡스톤 디자인2에서 구현
Login Server		해당사항 없음
Asset Server		해당사항 없음
Room Server		해당사항 없음
Game Server		해당사항 없음
버전관리		해당사항 없음

5. 요구사항을 충족시키지 못한 성능, 품질 요구사항이 있다면 그 이유와 해결방안을 기술하시오.

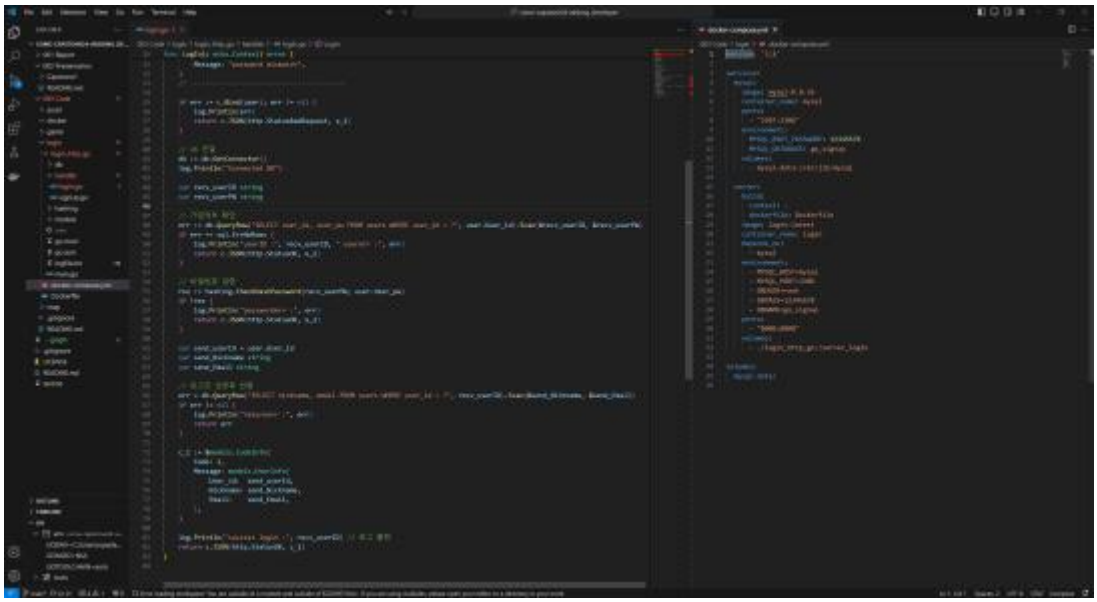
분류(성능, 속도 등) 및 최초 요구사항	충족 여부(현재 측정결과 제시)	이유(일정부족, 프로젝트 관리미비, 팀원변동, 기술적 문제 등)
확장성(스케일링)	미구현	캡스톤 디자인2에서 구현
네트워킹(통신)	미구현	캡스톤 디자인2에서 구현

6. 최종 완성된 프로젝트 결과물(소프트웨어, 하드웨어 등)을 설치하여 사용하기 위한 사용자 매뉴얼을 작성하시오.

해당 링크를 통해 소스 코드를 다운 받아 사용할 수 있습니다.
: https://github.com/HBNU-SWUNIV/come-capstone24-akdong_developer

마. Game Server

1) Login Server

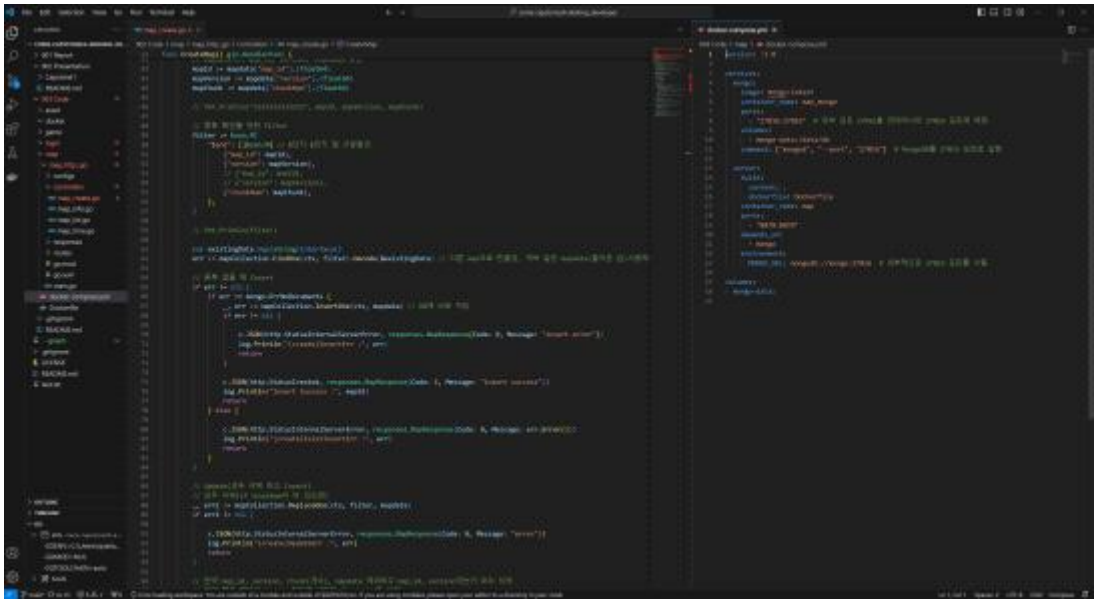


```
$ docker-compose up          // docker-compose 실행
```

(POST) ip:8000/signup // 사용자 회원가입 요청

(POST) ip:8000/login // 사용자 로그인 요청

2) Map Server



$ docker-compose up // docker-compose 실행

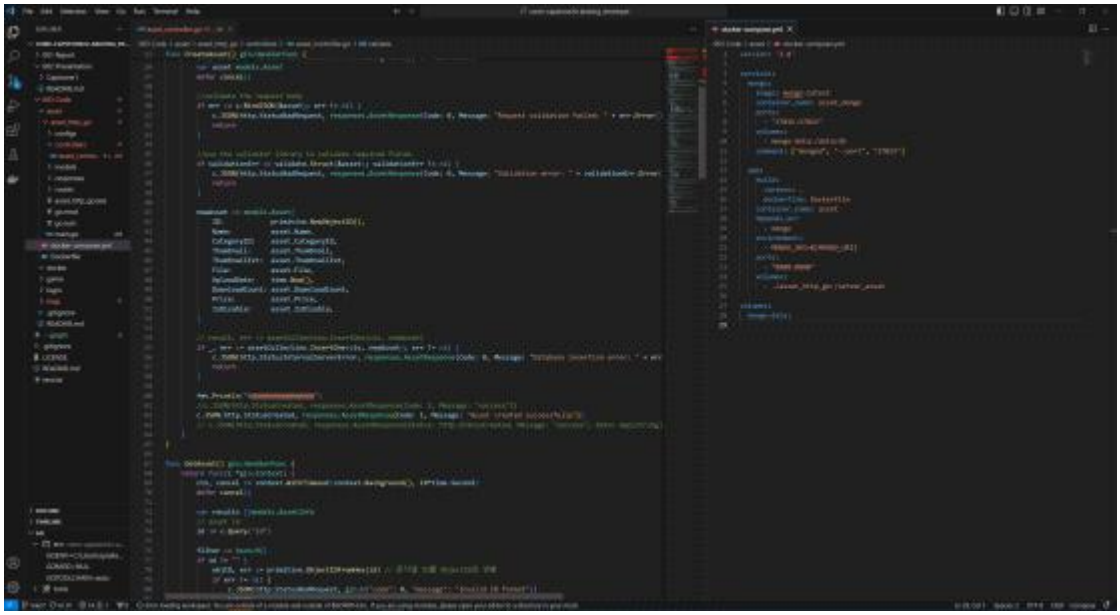
(POST) ip:8070/map_data // mapdata 저장

(GET) ip:8070/map_data?mapID="mapID"&version="version"&chunk="chunk"

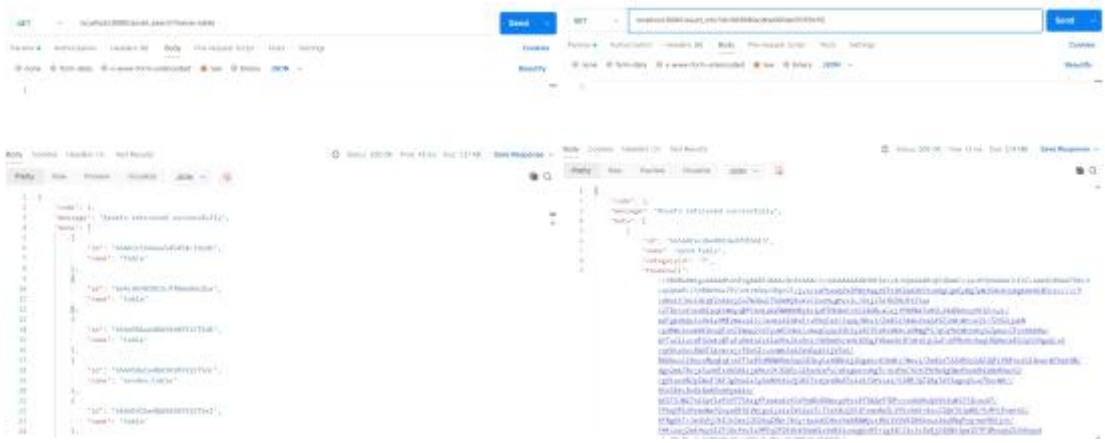
(GET) ip:8070/maplist // maplist 요청

(GET) ip:8070/maptime?mapId="mapId" // maptime 요청

3) Asset Server



```
$ docker-compose up            // docker-compose 실행
```



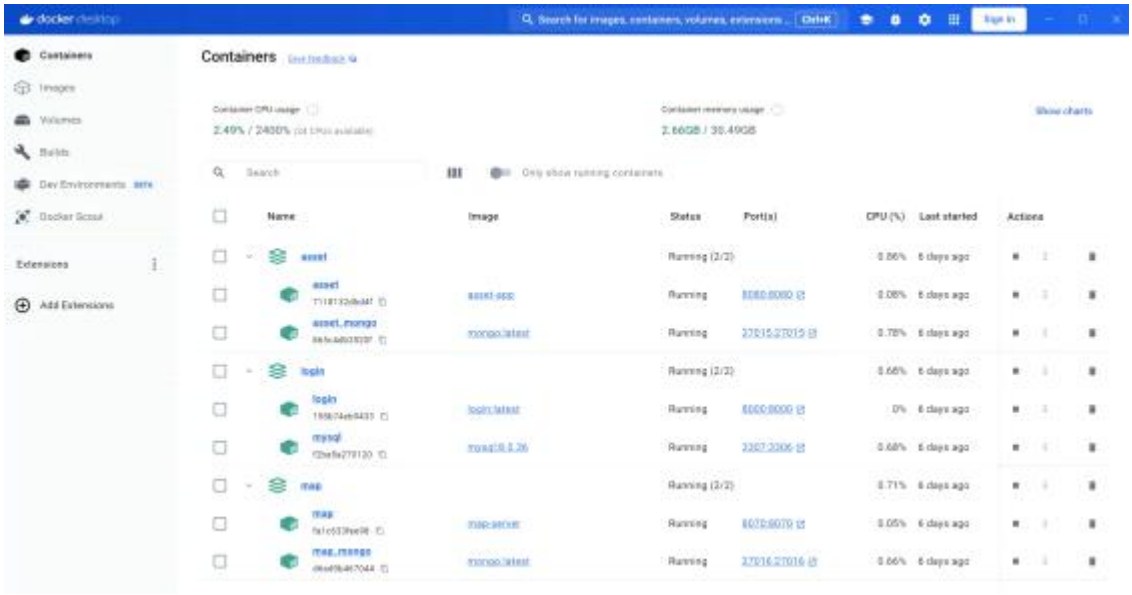
(POST) ip:8080/asset_upload   // assset 업로드 요청

(GET) ip:8080/asset_search?name="name"&categoryid="categhoryid"

(GET) ip:8080/asset_info&id="assetID"

(GET) ip:8080/asset_down?id="assetID"

4) Docker 실행 (인터페이스)



바. Image Build

```
$ mkdir -p ~/bin
```

```
$ cp Carte ~/bin/
```

```
$ echo 'export PATH=$PATH:~/bin' >> ~/.bashrc
```

```
$ source ~/.bashrc
```

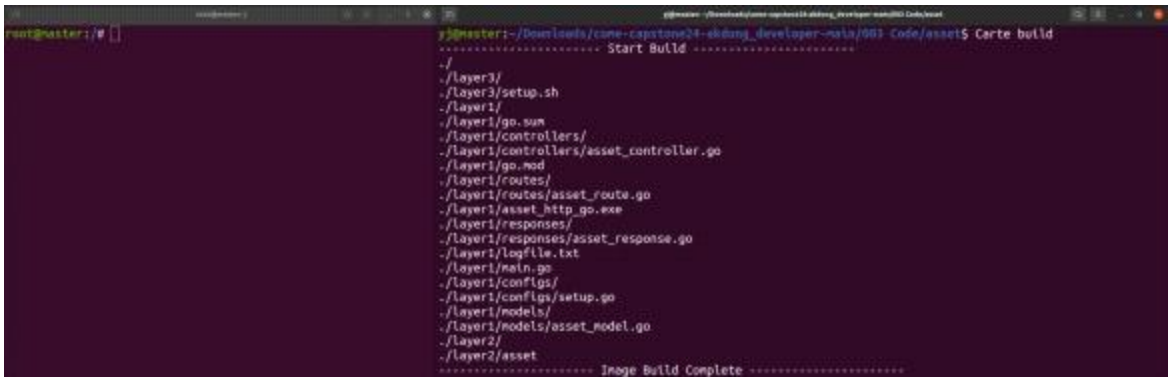
```
$ sudo mkdir /Carte
```

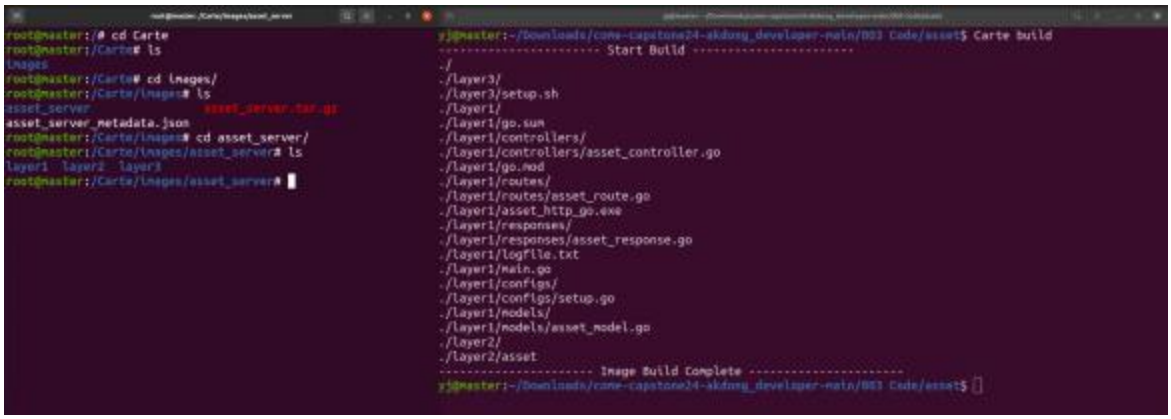
```
$ sudo mkdir /Carte/images
```

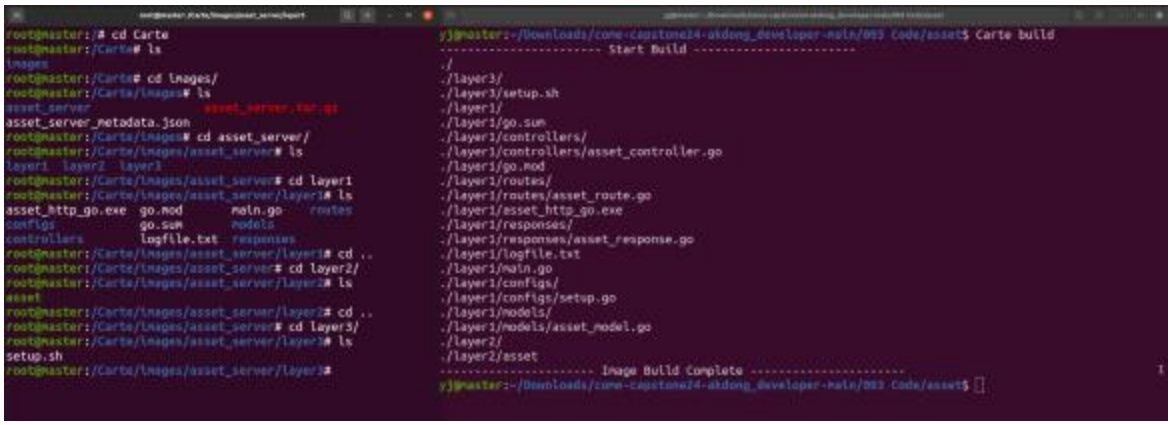
```
$ whoami
```

```
$ sudo chown -R yj(whoami) /Carte/images
```


root@master:/#
yj@master:~/Downloads/core-capstone24-akdong_developer-main/003 Code/asset$ Carte build


root@master:/#
yj@master:~/Downloads/core-capstone24-akdong_developer-main/003 Code/asset$ Carte build
---------- Start Build ----------
./
./layer3/
./layer3/setup.sh
./layer1/
./layer1/go.sum
./layer1/controllers/
./layer1/controllers/asset_controller.go
./layer1/go.mod
./layer1/routes/
./layer1/routes/asset_route.go
./layer1/asset_http_go.exe
./layer1/responses/
./layer1/responses/asset_response.go
./layer1/logfile.txt
./layer1/main.go
./layer1/configs/
./layer1/configs/setup.go
./layer1/models/
./layer1/models/asset_model.go
./layer2/
./layer2/asset
---------- Image Build Complete ----------


root@master:/# cd Carte
root@master:/Carte# ls
images
root@master:/Carte# cd images/
root@master:/Carte/images# ls
asset_server asset_server.tar.gz
asset_server_metadata.json
root@master:/Carte/images# cd asset_server/
root@master:/Carte/images/asset_server# ls
layer1 layer2 layer3
root@master:/Carte/images/asset_server#
yj@master:~/Downloads/core-capstone24-akdong_developer-main/003 Code/asset$ Carte build
---------- Start Build ----------
./
./layer3/
./layer3/setup.sh
./layer1/
./layer1/go.sum
./layer1/controllers/
./layer1/controllers/asset_controller.go
./layer1/go.mod
./layer1/routes/
./layer1/routes/asset_route.go
./layer1/asset_http_go.exe
./layer1/responses/
./layer1/responses/asset_response.go
./layer1/logfile.txt
./layer1/main.go
./layer1/configs/
./layer1/configs/setup.go
./layer1/models/
./layer1/models/asset_model.go
./layer2/
./layer2/asset
---------- Image Build Complete ----------
yj@master:~/Downloads/core-capstone24-akdong_developer-main/003 Code/asset$


root@master:/# cd Carte
root@master:/Carte# ls
images
root@master:/Carte# cd images/
root@master:/Carte/images# ls
asset_server asset_server.tar.gz
asset_server_metadata.json
root@master:/Carte/images# cd asset_server/
root@master:/Carte/images/asset_server# ls
layer1 layer2 layer3
root@master:/Carte/images/asset_server# cd layer1
root@master:/Carte/images/asset_server/layer1# ls
asset_http_go.exe go.mod main.go routes
configs go.sum models
controllers logfile.txt responses
root@master:/Carte/images/asset_server/layer1# cd ..
root@master:/Carte/images/asset_server# cd layer2/
root@master:/Carte/images/asset_server/layer2# ls
asset
root@master:/Carte/images/asset_server/layer2# cd ..
root@master:/Carte/images/asset_server# cd layer3/
root@master:/Carte/images/asset_server/layer3# ls
setup.sh
root@master:/Carte/images/asset_server/layer3#
yj@master:~/Downloads/core-capstone24-akdong_developer-main/003 Code/asset$ Carte build
---------- Start Build ----------
./
./layer3/
./layer3/setup.sh
./layer1/
./layer1/go.sum
./layer1/controllers/
./layer1/controllers/asset_controller.go
./layer1/go.mod
./layer1/routes/
./layer1/routes/asset_route.go
./layer1/asset_http_go.exe
./layer1/responses/
./layer1/responses/asset_response.go
./layer1/logfile.txt
./layer1/main.go
./layer1/configs/
./layer1/configs/setup.go
./layer1/models/
./layer1/models/asset_model.go
./layer2/
./layer2/asset
---------- Image Build Complete ----------
yj@master:~/Downloads/core-capstone24-akdong_developer-main/003 Code/asset$

7. 캡스톤디자인 결과의 활용방안

가. 기대효과

1) 효율적 서버 운영과 비용 절감

본 컨테이너 오케스트레이션 플랫폼의 서버 배포, 관리 및 확장의 자동화로 개발자들은 유연하고 경제적으로 서버 운영이 가능할 것이다, 또한 서버 배포 시간이 단축으로 비용 절감 및 시장 출시 기간을 줄일 것으로 기대된다.

2) 사용자 친화적 관리 인터페이스

직관적인 GUI 기반 관리 도구를 제공함으로써, 비전문가도 쉽게 시스템을 모니터링하고 관리할 수 있다. 이는 개발자의 작업 부담을 줄이고, 생산성을 향상시킬 것으로 기대된다.

3) 지적 재산권 확보

특허 및 실용신안 출원을 통해 개발된 기술의 지적 재산권을 확보함으로써, 기술적 리더십을 확립하고 장기적인 사업화 기반을 마련할 수 있다,

나. 활용방안

1) 클라우드 애플리케이션 배포 및 관리

다양한 클라우드 환경에서 애플리케이션의 배포와 관리를 단순화하고 최적화하여, 클라우드 기반 서비스 제공업체뿐만 아니라 다양한 산업 분야에서 광범위하게 활용될 수 있다.

2) 교육 분야에서의 활용

컨테이너화 및 오케스트레이션 기술 교육 자료로 활용될 수 있으며, 실습 중심의 학습을 통해 학생들에게 현대적인 인프라 관리 기술을 가르치는 데 유용한 도구가 될 수 있다,

본 과제의 결과물은 기술적, 경제적 가치를 넘어 사회적 영향력을 발휘할 수 있는 다양한 활용 가능성을 내포하고 있다. 특히, 컨테이너 오케스트레이션 툴을 통해 복잡한 인프라 관리의 문제를 해결하고자 하는 기업이나 개발자들에게 큰 도움을 줄 것으로 기대된다.