캡스톤디자인I 중간보고서

<table>
<tr><td rowspan="2">제 목</td><td>국문</td><td colspan="3">백엔드 서비스 프로바이더:컨테이너 오케스트레이션 플랫폼 개발</td></tr>
<tr><td>영문</td><td colspan="3">Backend Service Providers: Container Orchestration Platform Development</td></tr>
<tr><td rowspan="2">진 행
상 황</td><td>중요마일스톤</td><td colspan="3">기능 요구사항에 대한 중요 마일스톤
1) 컨테이너 런타임 관리 - 컨테이너 상태를 예측하고 파악하며 자원 상태 설정 필요
2) 네트워킹 - 컨테이너 외부 및 내부 통신에 대한 설계서
3) 스케일링 - CPU, 자원 상태 파악 후 특정 조건에 따른 자동 스케일링
(기능 요구사항에 대해 자동화된 소스코드 구현 필요)
테스트 요구사항에 대한 중요 마일스톤
1) Login Server, Room Server, Asset Server : Http통신
2) Game Server : Tcp/Udp통신
프로젝트 관리 요구사항에 대한 중요 마일스톤
1) 버전 관리 - Github을 통한 프로젝트 자산의 버전 관리
2) 협업 작업 - 다중 브랜치 및 병합 기능 지원
3) 협업 도구 활용 - Notion, Teams 활용
(협업 도구 활용 및 Github 관리를 통한 프로젝트 기능 수행)</td></tr>
<tr><td>진행상황</td><td colspan="3">기능 요구사항
기능 요구사항에 대한 시스템 구성도 작성 완료
기능 요구사항 - 리눅스 기반 컨테이너 생성 완료
기능 요구사항 - 컨테이너간 1:1 통신 구현 완료
2개의 컨테이너에 대한 연결 자동화 코드 구현 진행 중
테스트 요구사항
테스트 요구사항에 대한 구조도 작성 및 계획 수립 완료
테스트 요구사항에 대한 데이터베이스 구조 작성 완료
테스트 요구사항 - Login Server 구현 완료
테스트 요구사항 - Map Server, Asset Server 구현 마무리 작업 중</td></tr>
<tr><td>산출물</td><td colspan="4">요구사항 정의서(별첨 1), 중간보고서(별첨 2)</td></tr>
<tr><td rowspan="3">팀
구성원</td><td>학년</td><td>학 번</td><td>이 름</td><td>연락처(전화번호/이메일)</td></tr>
<tr><td>4</td><td>20211939</td><td>허유정</td><td>01022352464/20211939@edu.hanbat.ac.kr</td></tr>
<tr><td>4</td><td>20217140</td><td>김창인</td><td>01071561720/20217140@edu.hanbat.ac.kr</td></tr>
<tr><td colspan="5">컴퓨터공학과의 프로젝트 관리규정에 따라 다음과 같이 요구사항 정의서와 중간보고서를 제출합니다

2024 년 05 월 01 일

책임자 : 허유정(인)
지도교수 : 최창범(인)</td></tr>
</table>

[별첨1]

프로젝트명 : 백엔드 서비스 프로바이더:컨테이너 오케스트레이션 플랫폼 개발

소프트웨어 요구사항 정의서

Version 1.0

개발 팀원 명(팀리더):허유정
김창인

대표 연락처:010-2235-2464
e-mail: 20211939@edu.hanbat.ac.kr

목차

1. 시스템 개요

가. 사업의 목표

1) 컨테이너 오케스트레이션 플랫폼 제작

가) 컨테이너 오케스트레이션

현대의 디지털 환경은 다양하고 복잡한 워크로드를 신속하고 효율적으로 처리할 수 있는 능력을 요구한다. 이에 애플리케이션을 빠르게 배포하고 확장할 수 있는 수단인 컨테이너 기술이 등장하였으며 애플리케이션과 의존성을 패키징하여 소프트웨어를 일관된 환경에서 실행할 수 있는 서버 관리의 효율성이 증대되었다.

나) 컨테이너 오케스트레이션의 등장

서버 컨테이너 수가 증가함에 따라 대규모로 컨테이너를 배포하고 관리하는 과정에서 복잡성을 겪게 되었으며 이를 관리하기 위해 컨테이너 오케스트레이션 도구의 필요성이 대두되었다. 컨테이너 오케스트레이션은 컨테이너의 배포, 관리, 확장 등을 자동화함으로써, 복잡한 컨테이너 환경을 효율적으로 운영할 수 있다.

다) 컨테이너 오케스트레이션 도구의 현재 상황과 한계

쿠버네티스는 컨테이너 오케스트레이션의 대표적인 도구로 확장성, 유연성, 자동화의 장점을 지닌다. 언어나 프레임 워크에 구애받지 않아 개발자와 운영자의 작업 부담을 줄여주지만 구조가 복잡하며 높은 학습 곡선으로 소규모 프로젝트나 조직에 오버엔지니어링이 된다는 문제가 있다.

라) 플랫폼 제작 목표

쿠버네티스의 복잡성을 해결하고 보다 간편한 사용성을 제공하는 컨테이너 오케스트레이션 도구를 개발하고자 한다. 모니터링 서비스를 적용하여 관리하기 용이하며 특정 제품에 종속되지 않는 플랫폼을 제작으로 다양한 규모의 조직이 효율적인 관리 및 운영이 가능하도록 한다. 또한 게임 서버 구축으로 간편한 배포, 관리, 확장이 가능한지 실용성 여부를 확인해보고자 한다.

2) 멀티 사용자 관리 서버 구축

가) 게임 서버 구축 목표

개발된 컨테이너 오케스트레이션 플랫폼의 기능성을 검증하기 위해 실시간 멀티플레이어 게임 서버를 구축하고 테스트한다. 이를 통해 플랫폼의 성능, 안정성, 그리고 확장성을 평가하고, 최종적으로 고성능, 고가용성을 보장하는 게임 서버 인프라를 구현한다.

나. 추진 범위

1) 컨테이너 오케스트레이션

가) 사용 기술

본 사업은 컨테이너화된 애플리케이션의 배포, 관리 및 확장을 자동화하는 컨테이너 오케스트레이션 플랫폼 구축을 목표로 한다. 프로젝트는 Go 및 Rust 프로그래밍 언어를 사용하여 개발될 예정이며, Linux 운영체제 상에서 실행될 것이다. 이 플랫폼은 사용자 친화적인 웹 기반 사용자 인터페이스를 통해, 사용자가 효율적으로 컨테이너를 관리하고 모니터링 할 수 있도록 한다.

나) 구현 기능

플랫폼의 핵심 기능인 컨테이너 런타임 관리, 스케일링, 네트워킹, 사용자 인터페이스 관리로 사용자는 컨테이너의 라이프사이클을 쉽게 제어하고, 필요에 따라 리소스를 동적으로 조정 가능하다. 또한, 복잡한 네트워크 설정과 통신 규칙을 간편하게 구성할 수 있는 기능을 제공함으로써, 컨테이너 간 및 외부 네트워크와의 연결을 용이하게 할 수 있다.

다. 시스템 구성도

1) 컨테이너 오케스트레이션

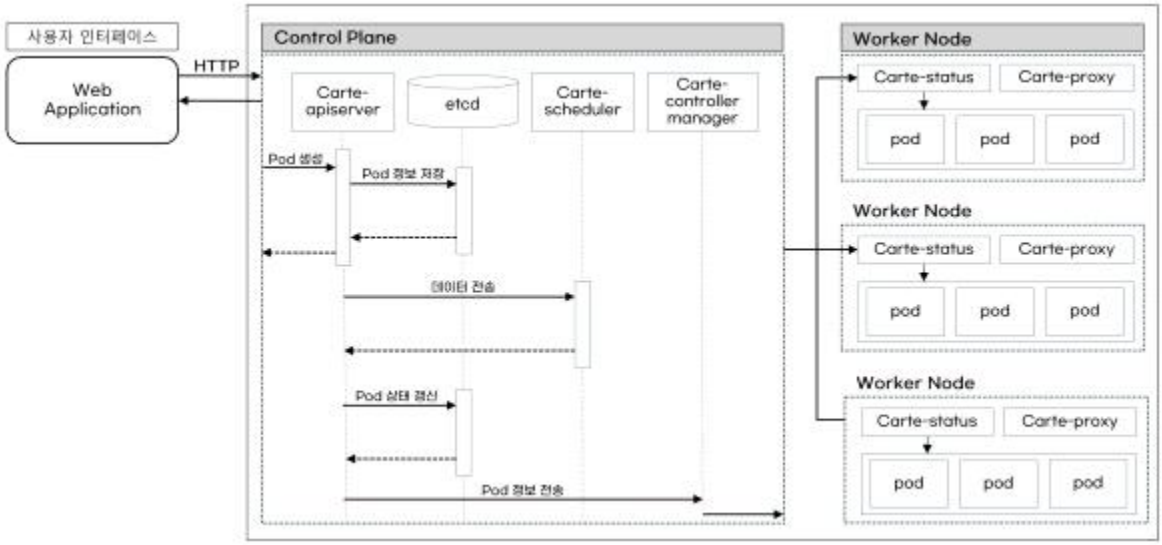


그림 1 시스템 구성도(컨테이너 오케스트레이션 플랫폼)

2) 게임 서버

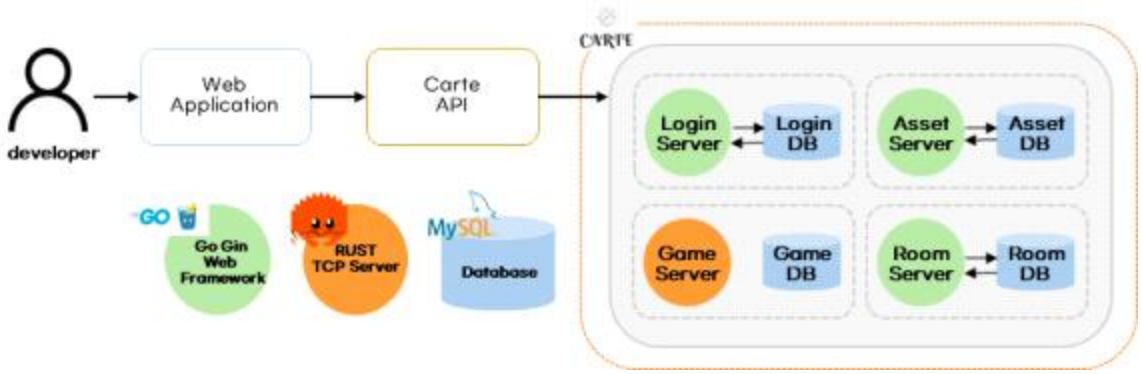


그림 1 게임 서버 구조도

2. 기능 요구사항

요구사항 고유번호		SFR-001		
요구사항 명칭		컨테이너 런타임 관리		
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		네트워킹		
요구사항 분류		기능	응락수준	필수
요구사항 상세설명	정의	컨테이너 간 통신 및 외부 네트워크와의 연결을 관리 할 수 있는 기능		
	세부내용	① 컨테이너 간의 통신을 위한 네트워킹 기능 ② 네트워크 구성 확인 및 조정 가능 ③ 가상 네트워크 환경 구성 ④ 네트워크 대역폭 및 성능 관리 기능 ⑤ (부가 요소) 보안 및 액세스 제어		

요구사항 고유번호		SFR-002		
요구사항 명칭		스케일링		
요구사항 분류		기능	응락수준	필수
요구사항 상세설명	정의	수동 또는 자동으로 컨테이너의 인스턴스 수를 조정 가능한 기능		
	세부내용	① 컨테이너 클러스터 크기 동적 조절 기능 ② 스케일링 정책 설정에 맞춘 자동 스케일링 작업 기능 ③ 자원의 효율성과 성능을 고려한 컨테이너 배치 가능 ④ 로드밸런싱 기능을 활용한 가용성 보장 ⑤ 스케일링 활동에 대한 시스템 상태 추적 가능		

3. 인터페이스 요구사항

요구사항 고유번호		SIR-001		
요구사항 명칭		모니터링		
요구사항 분류		인터페이스	응락수준	필수
요구사항 상세설명	정의	컨테이너의 CPU 및 메모리 사용량을 실시간으로 모니터링 가능하다		
	세부내용	① 네트워크 트래픽 및 I/O활동 모니터링 기능 ② 데이터 시각적 표현 기능 ③ 대시보드 형태 제공으로 직관적인 파악 가능 ④ 이상 징후 탐지 및 경고 알림 메시지를 통한 기능 포함		

요구사항 고유번호		SIR-002		
요구사항 명칭		컨테이너 동작 조절		
요구사항 분류		인터페이스	응락수준	필수
요구사항 상세설명	정의	인터페이스를 통한 컨테이너 시작, 중지, 재시작 기능 제공		
	세부내용	① 컨테이너 상태 변경 이력 기록 및 추적 로그 파악 가능 ② 컨테이너 동작 상태 실시간 모니터링 ③ 권한이 있는 사용자에게만 해당 작업 수행		

4. 보안 요구사항

요구사항 고유번호	SER-001		
요구사항 명칭	데이터 보안		
요구사항 분류	보안	응락수준	선택
요구사항 세부내용	① 컨테이너화된 애플리케이션에서 사용되는 데이터 암호화 지원 ② 데이터의 이동과 저장에 대한 액세스 제어를 구현하여 민감한 정보다 유출되지 않도록 보호 ③ 컨테이너 간 데이터 공유 시에도 데이터 무결성을 유지하기 위한 메커니즘 제공 ④ 적절한 데이터 백업 및 복구 기능 제공을 통한 데이터 손실 최소화 ⑤ 모니터링 기능과 함께 데이터 접근 및 사용 이력 추적 및 검증		

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		

5. 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	- 사용자 asset 정보 관리 기능 - 사용자 ID와 함께 asset 정보 관리 및 공개		

요구사항 고유번호	TER-003		
요구사항 명칭	Room 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		

6. 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		

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

요구사항 고유번호	PMR-003		
요구사항 명칭	작업 일정 관리		
요구사항 분류	프로젝트 관리	응락수준	선택
요구사항 세부내용	- 프로젝트의 작업 일정 관리 및 추적 기능 - 구글 시트를 통해 각 작업에 대한 시작일과 마감일 설정 - 진행 상황 실시간 업데이트를 통한 효율적인 프로젝트 관리		

[별첨2]

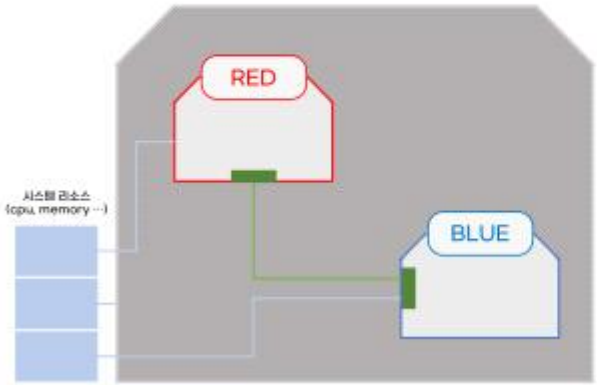
중간보고서

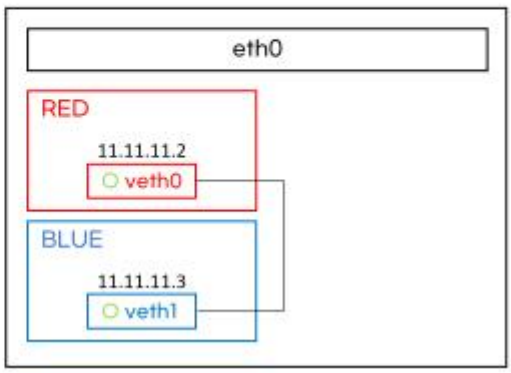
1. 요구사항 정의서에 명세된 기능에 대하여 현재까지 분석, 설계, 구현(소스코드 작성) 및 테스팅한 내용을 기술하시오.

[컨테이너 오케스트레이션] 도커 없이 컨테이너 생성

- 기존 기술을 활용하지 않고 리눅스 기반의 이미지 생성 및 네트워크 통신 과정을 살펴봄
- 한 서버에서 진행하였으며 두개의 컨테이너만을 생성

목표 : cgroup으로 자원 할당된 RED, BLUE 컨테이너 생성 후 통신 확인





이미지 사용

- (lower layer) 직접 생성한 tools, myroot 이미지 사용

```
tmp/tools
├── bin
│   ├── hostname
│   ├── ping
│   ├── rm
│   └── umount
├── lib
│   └── x86_64-linux-gnu
│       ├── libblkid.so.1
│       ├── libcap.so.2
│       ├── libc.so.6
│       ├── libdl.so.2
│       ├── libgcrypt.so.20
│       ├── libgpg-error.so.0
│       ├── libmount.so.1
│       ├── libm.so.6
│       ├── libpcre2-8.so.0
│       ├── libpthread.so.0
│       ├── libresolv.so.2
│       └── libselinux.so.1
├── lib64
│   └── ld-linux-x86-64.so.2
└── usr
    └── bin
        ├── stress
        └── which
6 directories, 19 files
```

```
tmp/myroot
├── bin
│   ├── ls
│   ├── mkdir
│   ├── mount
│   ├── ps
│   └── sh
├── escape_chroot
├── lib
│   └── x86_64-linux-gnu
│       ├── libblkid.so.1
│       ├── libc.so.6
│       ├── libdl.so.2
│       ├── libgcrypt.so.20
│       ├── libgpg-error.so.0
│       ├── liblz4.so.1
│       ├── liblzma.so.5
│       ├── libmount.so.1
│       ├── libpcre2-8.so.0
│       ├── libprocps.so.8
│       ├── libpthread.so.0
│       ├── librt.so.1
│       ├── libselinux.so.1
│       └── libsystemd.so.0
├── lib64
│   └── ld-linux-x86-64.so.2
├── proc
└── usr
    └── bin
        ├── ls
        ├── ps
        └── sh
8 directories, 23 files
```

cgroup 생성 및 설정

```
root@worker2:~# mkdir /sys/fs/cgroup/cpu/blue
root@worker2:~# mkdir /sys/fs/cgroup/memory/blue
root@worker2:~# echo 40000 > /sys/fs/cgroup/cpu/blue/cpu.cfs_quota_us;
root@worker2:~# echo 209715200 > /sys/fs/cgroup/memory/blue/memory.limit_in_bytes
root@worker2:~# echo 0 > /sys/fs/cgroup/memory/blue/memory.swappiness
```

- CPU : 40% 제한
- Memory : 200MB 제한 swap off (메모리 부족 시, 프로세스 종료)

컨테이너 격리 및 Cgroups 할당

mount / uts / ipc / fork pid

```
root@worker2:~# unshare -m -u -i -fp nsenter --net=/var/run/netns/BLUE /bin/sh;
# echo "1" > /sys/fs/cgroup/cpu/blue/cgroup.procs;
# echo "1" > /sys/fs/cgroup/memory/blue/cgroup.procs;
```

- 네트워크 : nsenter을 사용하여 var/run안의 BLUE 사용
- shell 실행

overlay mount

```
# mkdir /bluefs
# mkdir /bluefs/container
# mkdir /bluefs/work
# mkdir /bluefs/merge
# mount -t overlay overlay -o lowerdir=/tmp/tools:/tmp/myroot,upperdir=/bluefs/container,workdir=/bluefs/work /bluefs/merge
# tree /bluefs/merge
/bluefs/merge
├── bin
│   ├── hostname
│   ├── ls
│   ├── mkdir
│   ├── mount
│   ├── ping
│   ├── ps
│   ├── rm
│   ├── sh
│   └── umount
├── escape_chroot
├── lib
│   └── x86_64-linux-gnu
│       ├── libblkid.so.1
│       ├── libcap.so.2
│       ├── libc.so.6
│       ├── libdl.so.2
│       ├── libgcrypt.so.20
│       ├── libgpg-error.so.0
│       ├── liblz4.so.1
│       ├── liblzma.so.5
│       ├── libmount.so.1
│       ├── libm.so.6
│       ├── libpcre2-8.so.0
│       ├── libprocps.so.8
│       ├── libpthread.so.0
│       ├── libresolv.so.2
│       ├── librt.so.1
│       ├── libselinux.so.1
│       └── libsystemd.so.0
├── lib64
│   └── ld-linux-x86-64.so.2
├── proc
└── usr
    └── bin
        ├── ls
        ├── ps
        ├── sh
        ├── stress
        └── which
```

overlay mount를 위한 디렉토리

merge
container
tools
myroot

pivot_root

```
# mkdir -p /bluefs/merge/put_old
# cd /bluefs/merge
# pivot_root . put_old;
# cd /
# ls put_old
bin     boot   dev  home  lib32  libx32      media  opt   redfs  run   snap  swapfile  tmp  var
bluefs  cdrom  etc  lib   lib64  lost+found  mnt    proc  root   sbin  srv   sys       usr
# mount -t proc proc /proc;
# umount -l put_old
# rm -rf put_old
# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
0              1       0  0 06:31 ?        00:00:00 /bin/sh
0             18       1  0 06:37 ?        00:00:00 ps -ef
```

- 새로운 루트 파일 시스템 : overlay mount에서 생성함(merge: mount point)
- Mount point 진입 후, 현재 경로를 새로운 루트파일 시스템으로 놓고 기존 것을 put_old에 넣은 후 제거
  [put_old에는 현재 컨테이너가 속한 host의 root filesystem이 부착되어 있으므로 보안을 위해 제거함 : umount진행]

RED - BLUE 통신

```
# ping 11.11.11.3
PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data.
64 bytes from 11.11.11.3: icmp_seq=1 ttl=64 time=0.068 ms
64 bytes from 11.11.11.3: icmp_seq=2 ttl=64 time=0.047 ms
64 bytes from 11.11.11.3: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 11.11.11.3: icmp_seq=4 ttl=64 time=0.025 ms
64 bytes from 11.11.11.3: icmp_seq=5 ttl=64 time=0.051 ms
64 bytes from 11.11.11.3: icmp_seq=6 ttl=64 time=0.064 ms
64 bytes from 11.11.11.3: icmp_seq=7 ttl=64 time=0.056 ms
64 bytes from 11.11.11.3: icmp_seq=8 ttl=64 time=0.051 ms
64 bytes from 11.11.11.3: icmp_seq=9 ttl=64 time=0.046 ms
```

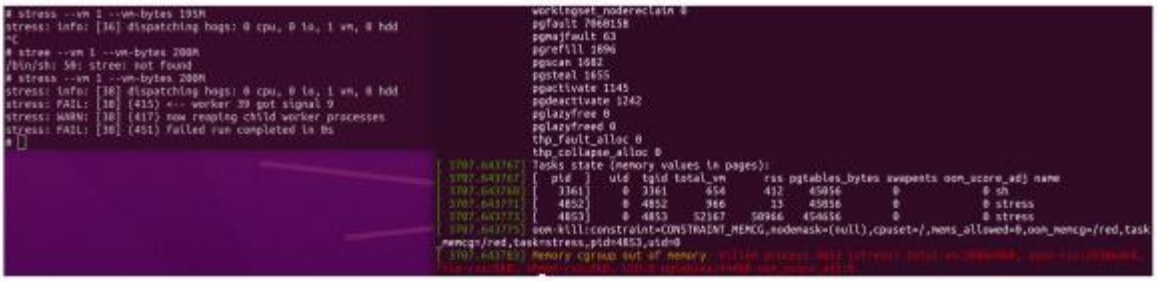
- RED -> BLUE

```
# ping 11.11.11.2
PING 11.11.11.2 (11.11.11.2) 56(84) bytes of data.
64 bytes from 11.11.11.2: icmp_seq=1 ttl=64 time=0.043 ms
64 bytes from 11.11.11.2: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 11.11.11.2: icmp_seq=3 ttl=64 time=0.056 ms
64 bytes from 11.11.11.2: icmp_seq=4 ttl=64 time=0.024 ms
64 bytes from 11.11.11.2: icmp_seq=5 ttl=64 time=0.046 ms
64 bytes from 11.11.11.2: icmp_seq=6 ttl=64 time=0.054 ms
64 bytes from 11.11.11.2: icmp_seq=7 ttl=64 time=0.059 ms
64 bytes from 11.11.11.2: icmp_seq=8 ttl=64 time=0.058 ms
64 bytes from 11.11.11.2: icmp_seq=9 ttl=64 time=0.047 ms
```

- BLUE -> RED

CPU 리소스, 메모리 리소스 확인

- CPU 리소스 확인



- 메모리 리소스 확인(200M 초과시 프로세스 종료)

[게임 서버] Login Server 구축
- 사용자 정보는 임의로 저장 후, 로그인 실행 정보 확인
- IP 주소를 통한 다른 컴퓨터에서 통신 가능

<main.go>

```
package main

import (
	"log"

	"module/bin"

	"capstone.com/module/handler"
	"github.com/joho/godotenv"
	"github.com/labstack/echo" // echo프레임워크 사용
)

func main() {

	err := godotenv.Load(".env")
	if err != nil {
		log.Fatal("Error loading .env file")
	}

	e := echo.New()
	e.POST("/signup", handler.SignUp)
	e.POST("/login", handler.LogIn)

	e.Logger.Fatal(e.Start(bin.IP))

}
```

<logIn.go>

```
package handler

import (
	"database/sql"
	"fmt"
	"net/http"

	"capstone.com/module/db"
	"capstone.com/module/hashing"
	"capstone.com/module/models"
	"github.com/labstack/echo"
)

func LogIn(c echo.Context) error {
	user := new(models.User)

	// model
	type UserInfo struct {
		// DB 고유 아이디 보내기
		User_id  string `json:"user_id"`
		Nickname string `json:"nickname"`
		Email    string `json:"email"`
	}

	type Code struct {
		Code    int    `json:"code"`
		Message string `json:"message"`
	}

	type CodeInfo struct {
		Code    int      `json:"code"`
		Message UserInfo `json:"message"`
	}

	// 전송 코드
	e_1 := &Code{
		Code:    0,
		Message: "bad request",
	}
```

```
	e_2 := &Code{
		Code:    0,
		Message: "user not found",
	}

	e_3 := &Code{
		Code:    0,
		Message: "password mismatch",
	}

	if err := c.Bind(user); err != nil {
		return c.JSON(http.StatusOK, e_1)
	}

	// db 연결
	db := db.GetConnector()
	fmt.Println("Connected DB")

	var recv_userID string
	var recv_userPW string
	// inputpw := user.User_pw

	// 가입여부 확인
	err := db.QueryRow("SELECT user_id, user_pw FROM users WHERE user_id =
?", user.User_id).Scan(&recv_userID, &recv_userPW)
	if err == sql.ErrNoRows {
		return c.JSON(http.StatusOK, e_2)
	}

	// 비밀번호 검증
	res := hashing.CheckHashPassword(recv_userPW, user.User_pw)
	if !res {
		return c.JSON(http.StatusOK, e_3)
	}

	var send_userId = user.User_id
	var send_Nickname string
	var send_Email string

	err2 := db.QueryRow("SELECT nickname, email FROM users WHERE user_id =
?", recv_userID).Scan(&send_Nickname, &send_Email)
	if err2 != nil {
```

```
            return err2
        }

        c_1 := &CodeInfo{
            Code: 1,
            Message: UserInfo{
                User_id:   send_userId,
                Nickname: send_Nickname,
                Email:     send_Email,
            },
        }

        return c.JSON(http.StatusOK, c_1)
}
```

<user.go>

```
package models

type User struct{
        User_id string `json:user_id`
        User_pw string `json:user_pw`
        Nickname string `json:nickname`
        Email string `json:email`
}
```

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

가. [컨테이너 오케스트레이션] 기존 기술 문제점 파악

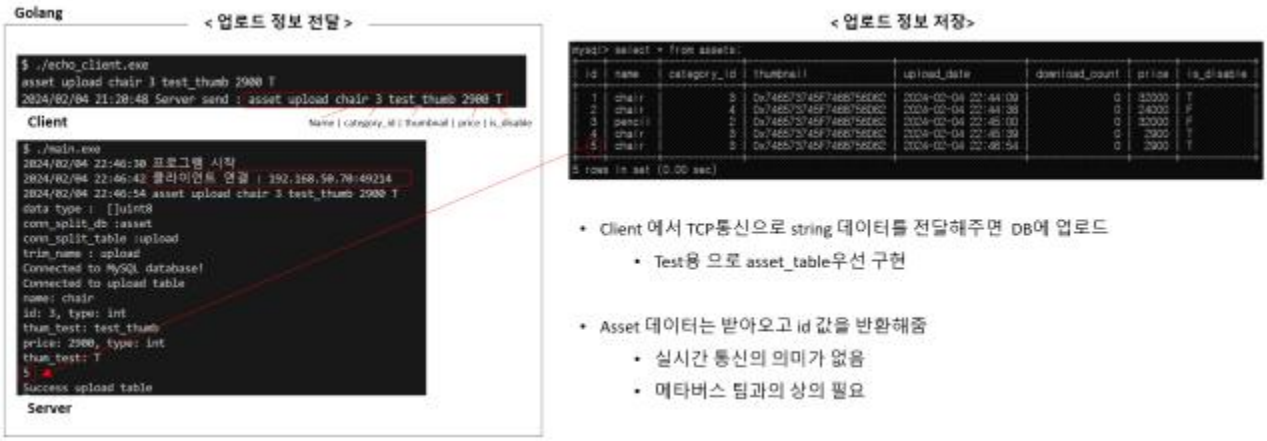
```
yj@master:~$ kubectl get node -o wide
NAME      STATUS   ROLES           AGE     VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-RUNTIME
master    Ready    control-plane   7m40s   v1.28.2   192.168.50.10    <none>        Ubuntu 20.04.6 LTS   5.15.0-91-generic   containerd://1.6.26
worker1   Ready    worker          5m48s   v1.28.2   192.168.50.88    <none>        Ubuntu 20.04.6 LTS   5.15.0-91-generic   containerd://1.6.26
worker2   Ready    worker          4m47s   v1.28.2   192.168.50.100   <none>        Ubuntu 20.04.6 LTS   5.15.0-91-generic   containerd://1.6.26
```

- 기존 기술인 쿠버네티스 서버 3개 연동을 통한 클러스트 구축 진행
- 학습곡선이 매우 높고 소규모의 프로젝트에서 사용하기 어렵다는 문제 발생
- 컨테이너 오케스트레이션 플랫폼 개발은 OCI를 적용한 기본 기능 구현을 목표로 설정

나. [컨테이너 오케스트레이션] 도커 없이 컨테이너 개발

- 경로 문제에 따른 (파일을 찾을 수 없는)오류 발생
- 명령어 형식에 따른 문제 발생

다. [게임 서버] Asset 정보 저장



- 형식을 정해서 보내는 것이 아닌 입력줄이 한줄로 들어감에 따른 문제 발생
- MySQL사용으로 데이터 복잡성 증가
- mongodb사용으로 더미 데이터 저장으로 변경

라. [게임서버] Golang 프레임워크 선정

- Go/gin : 가볍고 빠름, 미들웨어 사용으로 간편, 빠른 라우팅 지원
- Echo :높은 성능 제공의 경량 웹 프레임워크, 유연한 미들웨어 체인으로 요청, 응담
- Golang은 정해진 프레임워크가 존재하지 않아 각종 프레임워크의 특징 비교
- Go/gin을 처음에 사용하였지만 Echo프레임워크가 커스텀마이징이 용이하며, 중규모. 대규모 웹애플리케이션에 용이하여 선택

캡스톤 디자인 I 중간보고서 채점표

평가도구	평 가 항 목	평 가 점 수				
		1	2	3	4	5
중간 보고서 및 실행 결과	1. 요구사항 정의서(기능, 성능, 인터페이스 등)가 구체적으로 작성되었는가?					
	2. 요구분석, 설계 산출물(모델, 프로토타입 등)의 내용이 충실한가?					
	3. 설계 및 구현 문제를 위해 적용한 이론, 문제해결 방법이 제시되었으며 그 적용이 적합한가?					
	4. 구현된 소프트웨어(또는 이와 동등한 하드웨어 시스템)가 버그 없이 실행되었는가?					
	5. 구현된 소프트웨어(또는 이와 동등한 하드웨어 시스템)의 성능 요구사항은 충족되었는가?					
도구활용	6. 설계 및 구현을 위해 도구가 적절히 활용되었는가?					
	7. 도구의 활용수준(능숙도)은 프로젝트 수행에 적합한가?					
팀원의 업무 및 역할	8. 팀원의 업무분담에 따른 역할 및 협력이 충실히 이루어졌는가? (평가자에 의한 질의)					
	9. 프로젝트 중간 진척상황에 대해 팀원이 충분히 인지하고 있는가?(평가자에 의한 질의)					
합계						

***검토 의견(최종완료 때까지 보완해야할 점에 대해 작성 요망)**

심사위원(소속): **(이름)** **(인)**