

포팅 매뉴얼

LUMINA 프로젝트 포팅 매뉴얼

1. 배포 방식 소개

1.1. 수동 배포 방식

- EC2 인스턴스에서 Git Clone을 통해 소스 코드를 가져와 수동으로 배포.

1.2. CI/CD 방식

- 개발 및 운영에 사용한 Jenkins를 활용한 자동화된 파이프라인으로 배포.
-

2. 프로젝트 기술 스택

Frontend

- Visual Studio Code(IDE)
- HTML5, CSS3, JavaScript(ES2016)
- TypeScript: 5.7.2
- Node.js: 22.12.0
- React: 19.0.0
- Vite: 6.3.1
- TailwindCSS: 4.1.4
- Zustand: 5.0.3
- Google Analytics 4: 2.1.0

Backend

- IntelliJ(IDE): 2024.3.1.1

- Java: 21
- OpenJDK: 21
- Spring Boot: 3.4.4
- Gradle: 8.13.0
- MySQL: 8.4.5
- Redis: 7.4.3

AI

- Python: 3.12.6
- FastAPI: 0.115.12

ElizaOS

- TypeScript: 5.6.3
- Node.js: 23.3.0
- ElizaOS: 0.1.9

Infra

- GCP VM (개발 환경)
 - AWS EC2 (운영 환경)
 - AWS S3
 - Ubuntu: 22.04.4 LTS
 - Docker: 28.1.1
 - Docker Compose: 2.33.1
 - Jenkins: 2.492.3
 - Nginx: 1.26.3
 - Prometheus: 2.53.3
 - Grafana: 11.5.1
-

3. 외부 설정

3.1. 소셜 로그인 설정

- kakao Developer 설정
 - Redirect URI

```
http://localhost:5173/login/oauth2/code/kakao  
https://k12s306.p.ssafy.io/login/oauth2/code/kakao
```

- Google Cloud Platform 설정
 - Redirect URI

```
http://localhost:5173/login/oauth2/code/google  
https://k12s306.p.ssafy.io/login/oauth2/code/google
```

3.2. Google Analytics 설정

- 계정 및 속성을 만들기
- 데이터 스트림은 추후에 사용하는 도메인으로 설정
- 측정 ID는 복사하기 (frontend의 환경변수로 상용) (예: **G-ZJ4L01F08V**)

4. 수동 배포 방식 (Git Clone 기반)

4.1. 개요

이 섹션에서는 EC2 인스턴스에 접속하여 소스 코드를 클론하고, 수동으로 환경을 설정하여 애플리케이션을 배포하는 과정을 설명합니다.

4.2. 사전 준비

4.2.1. AWS의 EC2 인스턴스 생성

- 인스턴스 생성
 - 운영체제: Ubuntu (22.04 LTS 또는 24.04 LTS)
- PEM 키 생성 후 다운로드
- PEM키를 **C:\Users\[사용자 이름]\.ssh** 로 이동

4.2.2. 도메인 구매 (예: k12s306.p.ssafy.io)

- DNS 설정:
 - EC2 인스턴스의 공인 IP와 도메인을 연결

4.2.3. EC2 인스턴스 설정

- 로컬 PC의 바탕화면 우클릭 → 터미널에서 열기
 - EC2 인스턴스와 ssh 연결하는 명령어

```
ssh -i ~/.ssh/[PEM키 이름].PEM ubuntu@k12s306.p.ssafy.io
```

- 보안 그룹 및 방화벽 설정
 - Uncomplicated FireWall(ufw)를 사용하여 아래 포트들을 열고 외부 접근 허용
 - 명령어

- ufw 활성화

```
sudo ufw enable
```

- ufw 비활성화

```
sudo ufw disable
```

- ufw 포트 허용

```
sudo ufw allow [포트 번호]
```

- ufw 포트 차단

```
sudo ufw deny [포트 번호]
```

- ufw 규칙 삭제

```
sudo ufw delete allow [허용한 포트 번호]  
sudo ufw delete deny [차단한 포트 번호]
```

- ufw 상태 확인

```
sudo ufw status
```

- ufw 규칙 초기화

```
sudo ufw reset
```

- ufw 규칙 변경 사항 적용

```
sudo ufw reload
```

- 포트 목록 및 용도:

- 22 : SSH 연결
- 80 : HTTP (Nginx)
- 443 : HTTPS (Nginx)
- 3001 : React - Blue
- 3002 : React - Green
- 8081 : Spring Boot - Blue
- 8082 : Spring Boot - Green
- 8001 : AI Server - Blue
- 8002 : AI Server - Green

4.2.4. 필수 소프트웨어 설치

- Git

- 패키지 목록 업데이트

```
sudo apt update
```

- Git 설치

```
sudo apt install -y git
```

- git 버전 확인

```
git --version
```

- Docker

- 패키지 목록 업데이트

```
sudo apt-get update
```

- 필수 유틸리티 설치

```
sudo apt-get install ca-certificates curl
```

- GPG 키를 저장하기 위한 디렉터리 설정

```
sudo install -m 0755 -d /etc/apt/keyrings
```

- Docker의 GPG 키를 다운로드

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg  
-o /etc/apt/keyrings/docker.asc
```

- GPG 키 파일에 읽기 권한 부여

```
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

- Docker 공식 저장소 추가

```
echo \  
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/  
keyrings/docker.asc] https://download.docker.com/linux/ubuntu \  
$(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_  
CODENAME}") stable" | \  
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

- 패키지 목록 업데이트

```
sudo apt-get update
```

- Docker 패키지 설치

```
sudo apt-get install docker-ce docker-ce-cli containerd.io  
docker-buildx-plugin docker-compose-plugin
```

4.3. 소스 코드 클론

- 프로젝트 소스 코드를 GitLab에서 클론합니다.

```
cd ~  
git clone [레포지토리 URL]  
cd [레포지토리 이름]
```

<참고>

```
cd /  
⇒ 루트 디렉토리로 이동
```

```
cd ~  
⇒ 사용자의 홈 디렉토리로 이동 (/home/[사용자 이름])
```

```
cd [디렉토리 이름]  
⇒ 하위 디렉토리로 이동
```

```
cd ../  
⇒ 상위 디렉토리로 이동
```

4.4. 환경 설정

4.4.1. Docker 네트워크 생성

- 먼저 Docker 네트워크를 생성합니다.

```
docker network create lumina-network
```

4.4.2. 초기 SSL 인증서 발급

- 처음에는 certbot을 standalone 모드로 실행하여 인증서를 발급 받습니다.

```
docker run --rm -it \  
-p 80:80 \  
-v /etc/letsencrypt:/etc/letsencrypt \  
-v /var/lib/letsencrypt:/var/lib/letsencrypt \  
-v /var/www/html:/var/www/html \  
certbot/certbot certonly --standalone \  
-d k12s306.p.ssafy.io --non-interactive --agree-tos \  
-m [이메일]
```

4.4.3. redis 설정

- redis 컨테이너의 비밀번호를 하드 코딩한 `redis.conf` 를 복사합니다. (git에는 올리지 않습니다)
 - `redis.conf` 파일이 필요한 위치로 이동

```
cd infra/prod
```

- `redis.conf` 의 내용 복사 (ctrl + c)

```
port 6379
```

```
tcp-backlog 511
```

```
timeout 0
```

```
tcp-keepalive 0
```

```
loglevel notice
```

```
logfile ""
```

```
syslog-enabled no
```

```
syslog-ident redis
```

```
databases 16
```



```
save 900 1
save 300 10
save 60 10000

stop-writes-on-bgsave-error yes

rdbcompression yes

rdbchecksum yes

dbfilename dump.rdb

dir ./

slave-serve-stale-data yes

slave-read-only yes

repl-diskless-sync no

repl-diskless-sync-delay 5

repl-disable-tcp-nodelay no

slave-priority 100

requirepass mysecretpassword

appendonly no

appendfilename "appendonly.aof"

appendfsync everysec

no-appendfsync-on-rewrite no

auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

```
aof-load-truncated yes

lua-time-limit 5000

slowlog-log-slower-than 10000

slowlog-max-len 128

latency-monitor-threshold 0

notify-keyspace-events ""

hash-max-ziplist-entries 512
hash-max-ziplist-value 64

list-max-ziplist-entries 512
list-max-ziplist-value 64

set-max-intset-entries 512

zset-max-ziplist-entries 128
zset-max-ziplist-value 64

hll-sparse-max-bytes 3000

activerehashing yes


client-output-buffer-limit normal 0 0 0
client-output-buffer-limit slave 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60

hz 10

aof-rewrite-incremental-fsync yes
```

- vi editor로 `redis.conf` 생성하기

```
vi redis.conf
```

- 내용 붙여넣기

```
ctrl + v
```

- 저장하고 vi editor 종료하기

```
:wq
```

4.4.4. 환경 변수 설정

- `infra/prod` 디렉토리에 `.env` 파일 생성 후 아래 내용을 복사, 붙여넣기

```
MYSQL_HOST=mysql
MYSQL_PORT=3306
MYSQL_DATABASE=lumina_db
MYSQL_USER=lumina_user
MYSQL_PASSWORD=mysecretpassword
MYSQL_ROOT_PASSWORD=rootpassword

REDIS_HOST=redis
REDIS_PORT=6379
REDIS_PASSWORD=mysecretpassword

GOOGLE_APP_ID=466537341147-ik7advq3s0v0t0r408ft0og6308c4u9d.
apps.googleusercontent.com
GOOGLE_REDIRECT_URL=https://k12s306.p.ssafy.io/login/oauth2/code/
google
GOOGLE_SECRET=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
KAKAO_APP_ID=da82b7a6d31dbb8bd34e9542d6fb5b46
KAKAO_REDIRECT_URL=https://k12s306.p.ssafy.io/login/oauth2/code/
kakao

JWT_ACCESS_EXP=600000
JWT_REDIS_EXP=86400
JWT_REFRESH_EXP=86400000
JWT_SECRET_KEY=secure_key_with_minimum_32_characters_length!
```

```
LOGIN_SUCCESS=https://k12s306.p.ssafy.io
```

```
AWS_ACCESS_KEY=AKIAZTX2R6WQKB3DAC7W
```

```
AWS_SECRET_KEY=FwoNIzz6ztEaBDZQMZ1EJZ4UKdRh5PZzBplJ6EK1
```

```
AWS_REGION=ap-northeast-2
```

```
AWS_S3_BUCKET=s3-lumina-bucket
```

```
LUMINA_POST=http://eliza:4000/api/v1/evaluate-post
```

```
LUMINA_COMMENT=http://eliza:4000/api/v1/reply-mention
```

```
AI_SERVER=https://k12s306.p.ssafy.io/ai-server/analyze
```

```
GEMMA_CATEGORY=http://43.200.21.116:8000/recommend
```

```
GEMMA_POST_CATEGORY=http://43.200.21.116:8000/post-categorize
```

- `infra/prod` 디렉토리에 `.env.front` 파일 생성 후 아래 내용을 복사, 붙여넣기

```
VITE_API_URL=https://k12s306.p.ssafy.io/api/v1
```

```
VITE_GA_MEASUREMENT_ID=G-ZJ4L01F08V
```

- `frontend` 디렉토리에 `.env.front` 파일 생성 후 아래 내용을 복사, 붙여넣기

```
VITE_API_URL=https://k12s306.p.ssafy.io/api/v1
```

```
VITE_GA_MEASUREMENT_ID=G-ZJ4L01F08V
```

- `luna` 디렉토리에 `.env` 파일 생성 후 아래 내용을 복사, 붙여넣기

```
# Required environment variables
```

```
DISCORD_APPLICATION_ID=1362309276055503019
```

```
DISCORD_API_TOKEN=MTM2MjMwOTI3NjA1NTUwMzAxOQ.GHNip1.  
yZaIES0YqLxUm9w0NJP4lwuYs7V8Alilo21ROg
```

```
DISCORD_PUBLIC_KEY=2cc664e52ebc4b718342dc5d58a6a7313da86  
e5762571114170160c9b8f68aed
```

```
DISCORD_CLIENT_ID=1362309276055503019
```

```
DISCORD_CLIENT_SECRET=n-gRSjsEeqKeZEQBOVCK7ZcSd_381U2l
```

```
DISCORD_ENABLED=true
```

```
DISCORD_BOT_TOKEN=MTM2MjMwOTI3NjA1NTUwMzAxOQ.GHNip1.
```

```
yZalES0YqLxUm9w0NJP4lwuYs7V8Alilo21ROg
```

```
LUNA_PLATFORM_PORT=4000 #외부 접근 포트번호(Luna API 서버 포트)
ELIZA_SERVER_PORT=3000 #내부 API 포트번호(ElizaOS 메인 서버 포트)
USE_LLM_TYPE=local # LLM 선택 (openrouter 또는 local)
LOCAL_LLM_ENDPOINT=http://43.200.21.116:8000/predict # Local
LLM 서버 엔드포인트
LOCAL_LLM_MODEL=Gemma # Local LLM 모델명
BACKEND_API_URL=https://k12s306.p.ssafy.io
TZ=Asia/Seoul
```

```
OPENAI_API_KEY=sk-proj-KMd_wNCqArant4LB6O2XCvVV1-GTIDJlqxuL
ToMWZtl_D5lozLt3MfhYg1SLTeqFi8YwNSdPsPT3BlbkFJcudyxCM4Z9o-X
t3SSfAsVEFOP-dzXkhl-KyVaZIkUz_rkHaRsPAb4NgFHkuhkpvuUmhU5OUv
sA # OpenAI API key, starting with sk-
OPENROUTER_API_KEY=sk-or-v1-b7d5a3dd43debabeaf80819b149e9ec
40f1444c1804153ee3501ee451a815212
```

```
# When true, disables interactive chat mode for background
process operation
DAEMON_PROCESS=false
```

4.5. 어플리케이션 실행

4.5.1. AI Agent 실행 (luna 디렉토리)

- `docker-compose.yaml` 파일 실행

```
docker compose up --build -d
```

4.5.2. 어플리케이션 실행 (infra/prod 디렉토리)

- `docker-compose.yml` 파일 실행

```
docker compose up --build -d
```

4.5.3. 모니터링 실행 (infra/prod/monitoring 디렉토리)

- `monitoring-compose.yml` 파일 실행

```
docker compose -f monitoring-compose.yml -p monitoring up -d
```

4.5.4. 프록시 실행 (`infra/prod/proxy` 디렉토리)

- `proxy-compose.yml` 파일 실행

```
docker compose -f proxy-compose.yml -p proxy up -d
```

4.6. GPU 서버 설정

4.6.1. AWS의 EC2 인스턴스 생성

- 인스턴스 생성
 - AMI: Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.6 (Ubuntu 22.04)
 - 인스턴스 유형: g6e.xlarge
- PEM 키 생성 후 다운로드
- PEM키를 `C:\Users\[사용자 이름]\.ssh` 로 이동

4.6.2. EC2 인스턴스 설정

- 로컬 PC의 바탕화면 우클릭 → 터미널에서 열기
 - EC2 인스턴스와 ssh 연결하는 명령어

```
ssh -i ~/.ssh/[PEM키 이름].PEM ubuntu@[GPU 서버 공인 IP]
```

- 보안 그룹 및 방화벽 설정
 - Uncomplicated FireWall(ufw)를 사용하여 아래 포트들을 열고 외부 접근 허용
 - 명령어

- ufw 활성화

```
sudo ufw enable
```

- ufw 포트 허용

```
sudo ufw allow [포트 번호]
```

- ufw 상태 확인

```
sudo ufw status
```

- ufw 규칙 변경 사항 적용

```
sudo ufw reload
```

- 포트 목록 및 용도:
 - **22** : SSH 연결
 - **8000** : AI사용을 위한 FastAPI 서버
- HuggingFace Access Token 설정
 - HuggingFace 로그인
 - **My Page → Access Tokens → Create new token**
 - 생성한 Access Token을 GPU 서버에 등록

```
echo 'export HUGGINGFACE_HUB_TOKEN=[생성한 Access Token]' >>
~/.bashrc
source ~/.bashrcexport HUGGINGFACE_HUB_TOKEN=[생성한 Access
Token]
```

4.6.3. 필수 소프트웨어 설치

- Git
 - 패키지 목록 업데이트

```
sudo apt update
```

- Git 설치

```
sudo apt install -y git
```

- git 버전 확인

```
git --version
```

4.6.4 소스 코드 클론

- 프로젝트 소스 코드를 GitLab에서 클론합니다.

```
cd ~  
git clone [레포지토리 URL]  
cd [레포지토리 이름]
```

4.6.5 GPU 서버 실행

- `gpu` 디렉토리에서 FastAPI 서버 실행

```
nohup python run.py &
```

4.7. 검증

- 컨테이너 상태 확인

```
docker ps
```

- 서비스 접근

```
https://k12s306.p.ssafy.io
```

- `prometheus`, `cadvisor` 접근 시

```
ID: admin  
PW: pass123#
```

5. CI/CD (Jenkins 기반)

5.1. 개요

이 섹션에서는 Jenkins를 활용한 CI/CD 파이프라인을 설정하고 배포하는 과정을 설명합니다.

5.2. 사전 준비

- 4.2 와 동일한 기본 환경 설정이 필요합니다.
 - 개발 서버와 운영 서버 2개 설정
 - 도메인
 - 개발 서버 (예: picscore.net)
 - 운영 서버 (예: k12s306.p.ssafy.io)

5.3. 환경 설정

5.3.1. Docker 네트워크 생성

- 4.4.1 과 동일하게 Docker 네트워크를 생성합니다.
 - 개발 서버와 운영 서버 모두

5.3.2. 초기 SSL 인증서 발급

- 4.4.2 와 동일하게 초기 SSL 인증서를 발급합니다.
 - 개발 서버와 운영 서버 모두

5.3.3. redis 설정

- redis 컨테이너의 비밀번호를 하드 코딩한 `redis.conf` 를 복사합니다. (git에는 올리지 않습니다)
 - `redis.conf` 파일이 필요한 디렉토리 생성 및 이동
 - 개발 서버

```
mkdir -p ~/lumina/infra/dev
```
 - 운영 서버

```
mkdir -p ~/lumina/infra/prod
```
 - `redis.conf` 파일 생성

port 6379

tcp-backlog 511

timeout 0

tcp-keepalive 0

loglevel notice

logfile ""

syslog-enabled no

syslog-ident redis

databases 16

save 900 1

save 300 10

save 60 10000

stop-writes-on-bgsave-error yes

rdbcompression yes

rdbchecksum yes

dbfilename dump.rdb

dir ./

slave-serve-stale-data yes

slave-read-only yes

repl-diskless-sync no

```
repl-diskless-sync-delay 5

repl-disable-tcp-nodelay no

slave-priority 100

requirepass mysecretpassword

appendonly no

appendfilename "appendonly.aof"

appendfsync everysec

no-appendfsync-on-rewrite no

auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb

aof-load-truncated yes

lua-time-limit 5000

slowlog-log-slower-than 10000

slowlog-max-len 128

latency-monitor-threshold 0

notify-keyspace-events ""

hash-max-ziplist-entries 512
hash-max-ziplist-value 64

list-max-ziplist-entries 512
list-max-ziplist-value 64

set-max-intset-entries 512
```

```

zset-max-ziplist-entries 128
zset-max-ziplist-value 64

hll-sparse-max-bytes 3000

activerehashing yes

client-output-buffer-limit normal 0 0 0
client-output-buffer-limit slave 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60

hz 10

aof-rewrite-incremental-fsync yes

```

5.3.4. 개발 서버 초기 디렉토리 설정 및 초기 실행

- 디렉토리 생성

```

mkdir -p ~/lumina/infra/dev/proxy
mkdir -p ~/lumina/infra/jenkins


```


- `~/lumina/infra/dev/proxy` 디렉토리에 `nginx.conf` 파일 생성

```

worker_processes auto;

events {
    worker_connections 1024;
}

http {
    #  MIME 타입 포함 및 기본 콘텐츠 타입 설정
    include    /etc/nginx/mime.types;
    default_type application/octet-stream;

    #  최대 요청 크기 설정

```

```

client_max_body_size 10M; # 10MB로 설정

# ✅ Docker DNS resolver 설정
resolver 127.0.0.11 valid=30s;

# ✅ 재시도 옵션 설정
proxy_connect_timeout 75s;
proxy_read_timeout 300s;
proxy_next_upstream error timeout http_500 http_502 http_503
http_504;

# ✅ HTTP 요청을 HTTPS로 강제 리디렉션
server {
    listen 80;
    server_name picscore.net;

    # ✅ Let's Encrypt 인증용 경로는 리디렉션 하지 않음
    location /.well-known/acme-challenge/ {
        allow all;
        root /var/www/html;
    }

    # ✅ 나머지 모든 요청은 HTTPS로 리디렉션
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    http2 on;
    server_name picscore.net;

    # ✅ SSL 인증서 설정
    ssl_certificate /etc/letsencrypt/live/picscore.net/full
chain.pem;
    ssl_certificate_key /etc/letsencrypt/live/picscore.net/
privkey.pem;

```

```

# ✅ SSL 설정 최적화
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers on;
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:
ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20
-POLY1305:ECDHE-RSA-CHACHA20-POLY1305;
ssl_session_cache shared:SSL:10m;
ssl_session_timeout 10m;

# ✅ HSTS 설정 (HTTPS 강제)
add_header Strict-Transport-Security "max-age=31536000;
includeSubDomains" always;

# ✅ 보안 헤더 추가
add_header X-Content-Type-Options nosniff;
add_header X-Frame-Options SAMEORIGIN;
add_header X-XSS-Protection "1; mode=block";

# ✅ Jenkins url
location /jenkins {
    set $jenkins_upstream "jenkins:8080";

    proxy_pass http://$jenkins_upstream;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forward
    ed_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_redirect http:// https://;
}
}
}

```

- ~/lumina/infra/dev/proxy 디렉토리에 proxy-compose.yml 파일 생성

```

services:
  nginx:
    image: nginx:1.26-alpine
    container_name: nginx
    restart: always
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - /etc/letsencrypt:/etc/letsencrypt:ro
      - /var/www/html:/var/www/html
    depends_on:
      - certbot
    networks:
      - lumina-network

  certbot:
    image: certbot/certbot
    container_name: certbot
    volumes:
      - /etc/letsencrypt:/etc/letsencrypt
      - /var/lib/letsencrypt:/var/lib/letsencrypt
      - /var/www/html:/var/www/html
    networks:
      - lumina-network
    # 초기 인증서 발급 후 자동 갱신 설정 (12시간마다 체크)
    entrypoint: "/bin/sh -c 'trap exit TERM; while ;; do certbot
    renew; sleep 12h & wait $$(!); done;'"

networks:
  lumina-network:
    external: true

```

- ~/lumina/infra/jenkins 디렉토리에 jenkins-compose.yml 파일 생성

```

services:
  jenkins:

```

```

image: jenkins/jenkins:lts-jdk21
container_name: jenkins
user: root
restart: always
environment:
  - TZ=Asia/Seoul
  - JAVA_OPTS=-Djenkins.model.JenkinsLocationConfiguration.url=https://picscore.net/jenkins -Djenkins.model.Jenkins.crumbsIssuerProxyCompatibility=true -Dhudson.model.DirectoryBrowserSupport.CSP=
  - JENKINS_OPTS=--prefix=/jenkins
volumes:
  - jenkins_home:/var/jenkins_home
  - /var/run/docker.sock:/var/run/docker.sock
  - /usr/bin/docker:/usr/bin/docker
networks:
  - lumina-network

```

```

volumes:
  jenkins_home:
    name: jenkins_home

```

```

networks:
  lumina-network:
    external: true

```

- `proxy-compose.yml` 파일 실행 (`lumina/prod/proxy` 디렉토리)

```
docker compose -f proxy-compose.yml -p proxy up -d
```

- `jenkins-compose.yml` 파일 실행 (`lumina/prod/jenkins` 디렉토리)

```
docker compose -f jenkins-compose.yml -p jenkins up -d
```

- 컨테이너 상태 확인

```
docker ps
```


5.3.4. Jenkins 설정

- Jenkins 초기 비밀번호 확인

```
docker exec -it jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

- Jenkins 웹 UI 접속
 - 브라우저에서 <https://picscore.net/jenkins> 접속 후, 초기 비밀번호를 입력합니다.
 - 그 후, 초기 설정을 진행합니다.
- 필수 플러그인 설치
 - **Jenkins 웹 UI → "Manage Jenkins" → "Manage Plugins"** 로 이동 후, 다음 플러그인들을 설치합니다.
 - **GitLab** → GitLab 웹훅 트리거 지원
 - **Pipeline** → Jenkinsfile을 실행하는 데 필요
 - **Docker Pipeline** → Docker와 연동하는 데 필요
 - **SSH Pipeline Steps** → EC2에 배포할 때 필요
 - **Credentials Binding** → Docker Hub 및 SSH 키 인증 필요
 - **SSH Agent** → 안전하게 SSH에 접속하기 위해 필요
 - **SonarQube Scanner for Jenkins** → SonarQube를 통해 코드를 분석하기 위해 필요
 - **NodeJS Plugin** → NodeJS를 사용하여 빌드하기 위해 필요
- GitLab Credentials 설정
 - GitLab의 Access Token 발급
 - **GitLab의 "User Settings" → "Access tokens" → "Add new token"**
 - **Token name** 입력
 - **Select scopes** 선택: `api`, `read_repository`, `write_repository` 선택
 - **"Create personal access token"**
 - Jenkins에 GitLab Access Token 등록
 - **Jenkins 웹 UI → "Manage Jenkins" → "Credentials" → "Domains (global)" → "Add Credentials"**

- **Kind:** Username with password
 - **Scope:** Global (Jenkins, nodes, items, all child items, etc)
 - **Username:** GitLab 아이디 입력
 - **Password:** GitLab의 Access Token 입력
 - **ID:** Credential의 이름(ID) 지정 (ex. gitlab-token)
 - **"Create"**
- Docker Hub Credentials 설정
 - Docker Hub Access Token 생성
 - Docker Hub 로그인
 - **"Account Settings" → "Security" → "Personal access tokens" → "Generate new token"**
 - **Access token description:** Access token 이름 입력
 - **Optional → Access permissions:** Read, Write, Delete 선택
 - **"Generate":** 생성 후 복사
 - Jenkins에서 Credentials 추가
 - **Jenkins 웹 UI → "Manage Jenkins" → "Domains의 (global)" → "Add Credentials"**
 - **Kind:** Username with password
 - **Scope:** Global (Jenkins, nodes, items, all child items, etc)
 - **Username:** Docker Hub 아이디 입력
 - **Password:** Docker Hub의 Access Token 입력
 - **ID:** Credential의 이름(ID) 지정 (ex. dockerhub-token)
 - **"Create"**
- SSH 키 Credentials 설정
 - EC2에서 SSH 키 생성 (개발 서버와 운영 서버 모두)

```
ssh-keygen -t rsa -b 4096 -C "jenkins"
```

 - 기본값으로 /home/ubuntu/.ssh/id_rsa 저장

- `id_rsa` → **Private Key** (Jenkins에서 사용)
- `id_rsa.pub` → **Public Key** (EC2 서버에 등록)

◦ EC2에 Public Key 추가

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

- `cat ~/.ssh/id_rsa.pub` → 현재 사용자의 **공개 키(public key)**를 출력.
- `>> ~/.ssh/authorized_keys` → 출력된 공개 키를 `~/.ssh/authorized_keys` 파일의 끝에 추가.

```
chmod 600 ~/.ssh/authorized_keys
```

- `chmod 600` → 해당 파일(`~/.ssh/authorized_keys`)의 권한을 보안 설정에 맞게 조정.
- `600` 권한:
 - 소유자(현재 사용자): 읽기(r), 쓰기(w).
 - 그룹 및 다른 사용자: 접근 불가.

◦ Jenkins에 Private Key 추가

- **Jenkins 웹 UI** → **"Manage Jenkins"** → **"Domains의 (global)"** → **"Add Credentials"**
- **Kind:** `SSH Username with private key`
- **Scope:** `Global (Jenkins, nodes, items, all child items, etc)`
- **ID:** Credential의 이름(ID) 지정 (예. 개발 서버: `gcp-ssh-key` , 운영 서버: `aws-ssh-key`)
- **Username:** `ubuntu`
- **Private Key** → **"Enter directly"** → **"Add":** `id_rsa` 파일 내용을 복사해서 붙여넣기
- **"Create"**

- `.env` 파일을 Credentials의 Secret File로 등록

◦ Jenkins에서 Secret File 등록

- **Jenkins 웹 UI 접속** → **"Manage Jenkins"** → **"Manage Credentials"** 이동

- **Domains**의 `(global)` → **"Add Credentials"** 선택
 - **Kind:** `Secret file` 선택
 - **Scope:** `Global (Jenkins, nodes, items, all child items, etc)` 선택
 - **File:** `.env` 파일을 선택하여 업로드
 - **ID:** Pipeline에서 사용할 Credential ID 설정
 - **"Create"** 클릭하여 저장
- 필요한 `.env` 코드

- ID: `env-dev-content`

```

MYSQL_HOST=mysql
MYSQL_PORT=3306
MYSQL_DATABASE=lumina_db
MYSQL_USER=lumina_user
MYSQL_PASSWORD=mysecretpassword
MYSQL_ROOT_PASSWORD=rootpassword

REDIS_HOST=redis
REDIS_PORT=6379
REDIS_PASSWORD=mysecretpassword

GOOGLE_APP_ID=466537341147-ik7advq3s0v0t0r408ft0og6308
c4u9d.apps.googleusercontent.com
GOOGLE_REDIRECT_URL=https://picscore.net/login/oauth2/code/
google
GOOGLE_SECRET=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
KAKAO_APP_ID=da82b7a6d31dbb8bd34e9542d6fb5b46
KAKAO_REDIRECT_URL=https://picscore.net/login/oauth2/code/
kakao

JWT_ACCESS_EXP=600000
JWT_REDIS_EXP=86400
JWT_REFRESH_EXP=86400000
JWT_SECRET_KEY=secure_key_with_minimum_32_characters
_length!

```

```
LOGIN_SUCCESS=https://picscore.net/
```

```
AWS_ACCESS_KEY=AKIAZTX2R6WQKB3DAC7W
```

```
AWS_SECRET_KEY=FwoNlzz6ztEaBDZQMZ1EJZ4UKdRh5PZZBplJ  
6EK1
```

```
AWS_REGION=ap-northeast-2
```

```
AWS_S3_BUCKET=s3-lumina-bucket
```

```
LUMINA_POST=http://eliza:4000/api/v1/evaluate-post
```

```
LUMINA_COMMENT=http://eliza:4000/api/v1/reply-mention
```

```
AI_SERVER=https://picscore.net/ai-server/analyze
```

```
GEMMA_CATEGORY=http://43.200.21.116:8000/recommend
```

```
GEMMA_POST_CATEGORY=http://43.200.21.116:8000/  
post-categorize
```

■ ID: `env-front-dev-content`

```
VITE_API_URL=https://picscore.net/api/v1
```

```
VITE_GA_MEASUREMENT_ID=G-4NGWW4YN9L
```

■ ID: `htpasswd-dev-content`

```
admin:$apr1$NWN5DUeg$PjfeUwJyIrlGuRUwNtiNv/
```

■ ID: `env-luna-dev-content`

```
# Required environment variables
```

```
DISCORD_APPLICATION_ID=1362309276055503019
```

```
DISCORD_API_TOKEN=MTM2MjMwOTI3NjA1NTUwMzAxOQ.
```

```
GHNip1.yZaIES0YqLxUm9w0NJP4lwuYs7V8Alilo21ROg
```

```
DISCORD_PUBLIC_KEY=2cc664e52ebc4b718342dc5d58
```

```
a6a7313da86e576257114170160c9b8f68aed
```

```
DISCORD_CLIENT_ID=1362309276055503019
```

```
DISCORD_CLIENT_SECRET=n-gRSjsEeqKeZEqBOVck7ZcSd_  
381U2I
```

```
DISCORD_ENABLED=true
```

```
DISCORD_BOT_TOKEN=MTM2MjMwOTI3NjA1NTUwMzAxOQ.  
GHNip1.yZaIES0YqLxUm9w0NJP4lwuYs7V8Alilo21ROg
```

```
LUNA_PLATFORM_PORT=4000 #외부 접근 포트번호  
(Luna API 서버 포트)
```

```
ELIZA_SERVER_PORT=3000 #내부 API 포트번호  
(ElizaOS 메인 서버 포트)
```

```
USE_LLM_TYPE=local # LLM 선택 (openrouter 또는 local)  
LOCAL_LLM_ENDPOINT=http://43.200.21.116:8000/predict  
# Local LLM 서버 엔드포인트
```

```
LOCAL_LLM_MODEL=Gemma # Local LLM 모델명  
BACKEND_API_URL=https://picscore.net  
TZ=Asia/Seoul
```

```
OPENAI_API_KEY=sk-proj-KMd_wNCqArant4LB6O2XCvVV1-GTID  
JlqxuLT0MWZtl_D5lozLt3Mfhyg1SLTeqFi8YwNSdPsPT3BlbkFJc  
udyxCM4Z9o-Xt3SSfAsVEFOP-dzXkhl-KyVaZlkUz_rkHaRsPAb4N  
gFHkuhkpvuUmhU5OUvsA # OpenAI API key, starting with sk-  
OPENROUTER_API_KEY=sk-or-v1-b7d5a3dd43debabeaf80819b1  
49e9ec40f1444c1804153ee3501ee451a815212
```

```
# When true, disables interactive chat mode for background  
process operation  
DAEMON_PROCESS=false
```

■ ID: `env-prod-content`

```
MYSQL_HOST=mysql  
MYSQL_PORT=3306  
MYSQL_DATABASE=lumina_db  
MYSQL_USER=lumina_user  
MYSQL_PASSWORD=mysecretpassword  
MYSQL_ROOT_PASSWORD=rootpassword
```

```
REDIS_HOST=redis  
REDIS_PORT=6379  
REDIS_PASSWORD=mysecretpassword
```

GOOGLE_APP_ID=466537341147-ik7advq3s0v0t0r408ft0og6308
c4u9d.apps.googleusercontent.com
GOOGLE_REDIRECT_URL=https://k12s306.p.ssafy.io/login/oauth2/
code/google
GOOGLE_SECRET=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
KAKAO_APP_ID=da82b7a6d31dbb8bd34e9542d6fb5b46
KAKAO_REDIRECT_URL=https://k12s306.p.ssafy.io/login/oauth2/
code/kakao

JWT_ACCESS_EXP=600000
JWT_REDIS_EXP=86400
JWT_REFRESH_EXP=86400000
JWT_SECRET_KEY=secure_key_with_minimum_32_characters_
length!

LOGIN_SUCCESS=https://k12s306.p.ssafy.io

AWS_ACCESS_KEY=AKIAZTX2R6WQKB3DAC7W
AWS_SECRET_KEY=FwoNIzz6ztEaBDZQMZ1EJZ4UKdRh5PZzBpIJ
6EK1
AWS_REGION=ap-northeast-2
AWS_S3_BUCKET=s3-lumina-bucket

LUMINA_POST=http://eliza:4000/api/v1/evaluate-post
LUMINA_COMMENT=http://eliza:4000/api/v1/reply-mention

AI_SERVER=https://k12s306.p.ssafy.io/ai-server/analyze

GEMMA_CATEGORY=http://43.200.21.116:8000/recommend
GEMMA_POST_CATEGORY=http://43.200.21.116:8000/
post-categorize

- ID: env-front-prod-content

VITE_API_URL=https://k12s306.p.ssafy.io/api/v1
VITE_GA_MEASUREMENT_ID=G-ZJ4L01F08V

- ID: httpasswd-prod-content

```
admin:$apr1$NWN5DUeg$PjfeUwJyIrlGuRUwNtiNv/
```

- ID: `env-luna-prod-content`

```
# Required environment variables
DISCORD_APPLICATION_ID=1362309276055503019
DISCORD_API_TOKEN=MTM2MjMwOTI3NjA1NTUwMzAxOQ.
GHNip1.yZaIES0YqLxUm9w0NJP4lwuYs7V8Alilo21ROg
DISCORD_PUBLIC_KEY=2cc664e52ebc4b718342dc5d58
a6a7313da86e576257114170160c9b8f68aed
DISCORD_CLIENT_ID=1362309276055503019
DISCORD_CLIENT_SECRET=n-gRSjsEeqKeZEqBOVck7ZcSd_
381U2I
DISCORD_ENABLED=true
DISCORD_BOT_TOKEN=MTM2MjMwOTI3NjA1NTUwMzAxOQ.GHN
ip1.yZaIES0YqLxUm9w0NJP4lwuYs7V8Alilo21ROg

LUNA_PLATFORM_PORT=4000 #외부 접근 포트번호
(Luna API 서버 포트)
ELIZA_SERVER_PORT=3000 #내부 API 포트번호
(ElizaOS 메인 서버 포트)
USE_LLM_TYPE=local # LLM 선택 (openrouter 또는 local)
LOCAL_LLM_ENDPOINT=http://43.200.21.116:8000/predict # Local
LLM 서버 엔드포인트
LOCAL_LLM_MODEL=Gemma # Local LLM 모델명
BACKEND_API_URL=https://k12s306.p.ssafy.io
TZ=Asia/Seoul

OPENAI_API_KEY=sk-proj-KMd_wNCqArant4LB6O2XCvVV1-
GTIDJlqxuLToMWZtl_D5lozLt3Mfhgy1SLTeqFi8YwNSdPsPT
3BlbkFJcudyxCM4Z9o-Xt3SSfAsVEFOP-dzXkhl-KyVaZIkUz
_rkHaRsPAb4NgFHkuhkpvuumhU5OUvsA
# OpenAI API key, starting with sk-
OPENROUTER_API_KEY=sk-or-v1-b7d5a3dd43debabeaf80819b14
9e9ec40f1444c1804153ee3501ee451a815212

# When true, disables interactive chat mode for background
```



```
process operation
DAEMON_PROCESS=false
```

- ID: `sonarqube-token` (SonarQube에서 생성한 사용자 토큰)

```
sqa_1f83ff5f897ef65109a7964c136fd1e1c645a7b2
```

- SonarQube 설정
 - Jenkins 웹 UI 접속 → "Manage Jenkins" → "System" 이동
 - SonarQube servers 설정
 - Name: `SonarQube`
 - ServerURL: `http://sonarqube:9000`
 - server authentication token: `sonarqube-token`
 - Jenkins 웹 UI 접속 → "Manage Jenkins" → "Tools" 이동
 - NodeJS installations 설정
 - Name: `Node22`
 - Install automatically: `체크`
 - Version: `NodeJS 22.14.0`
 - Global npm packages to install: `npm install -g yarn`
- Multi-Branch Pipeline 구성 (1) ⇒ 기본 어플리케이션의 파이프라인
 - Multi-Branch Pipeline Job 생성
 - Jenkins 웹 UI → "New Item"
 - Job 이름 입력 → "Multi-Branch Pipeline" 선택 → "OK" (예: `Lumina`)
 - Branch Sources → `Git` 선택
 - Project Repository 입력: `GitLab 프로젝트 URL` 입력
 - Credentials 선택: `gitlab-token` 선택
 - Behaviours: `Discover branches`
 - Build Configuration: Script Path에 `infra/jenkins/Jenkinsfile` 입력
 - "Save"

- Jenkins에서 API 토큰 생성
 - Jenkins 웹 UI → "Manage Jenkins" → "Manage Users" → "admin"
 - "Security" → "API Token" 생성 후 복사
- GitLab에 웹훅 추가
 - GitLab 프로젝트로 이동
 - "Settings" → "Webhooks" → "Add New Webhook"
 - URL 입력: `https://admin:<jenkins-api-token>@picscore.net/jenkins/project/Lumina`
 - Trigger 선택: `Push events` , `Merge request events` 선택
 - "Add webhook"
- Multi-Branch Pipeline 구성 (2) ⇒ AI 에이전트의 파이프라인
 - Multi-Branch Pipeline Job 생성
 - Jenkins 웹 UI → "New Item"
 - Job 이름 입력 → "Multi-Branch Pipeline" 선택 → "OK" (예: `Luna`)
 - Branch Sources → `Git` 선택
 - Project Repository 입력: `GitLab 프로젝트 URL` 입력
 - Credentials 선택: `gitlab-token` 선택
 - Behaviours: `Discover branches`
 - Build Configuration: Script Path에 `luna/Jenkinsfile` 입력
 - "Save"
 - Jenkins에서 API 토큰 생성
 - Jenkins 웹 UI → "Manage Jenkins" → "Manage Users" → "admin"
 - "Security" → "API Token" 생성 후 복사
 - GitLab에 웹훅 추가
 - GitLab 프로젝트로 이동
 - "Settings" → "Webhooks" → "Add New Webhook"
 - URL 입력: `https://admin:<jenkins-api-token>@picscore.net/jenkins/project/Luna`
 - Trigger 선택: `Push events` , `Merge request events` 선택

- "Add webhook"

5.4. GPU 서버 설정

- 4.6 과 동일한 GPU 서버 설정이 필요합니다.

5.5. CI/CD 파이프라인 실행

5.5.1. AI 에이전트 실행

- Jenkins 웹 UI → "Luna"
- Branches에서 선택
 - 개발 서버: develop 브랜치 클릭
 - 운영 서버: master 브랜치 클릭
- 지금 빌드 클릭

5.5.2. 어플리케이션 실행

- Jenkins 웹 UI → "Lumina"
- Branches에서 선택
 - 개발 서버: develop 브랜치 클릭
 - 운영 서버: master 브랜치 클릭
- 지금 빌드 클릭

5.6. 검증

- 컨테이너 상태 확인

```
docker ps
```

- 서비스 접근
 - 개발 서버

```
https://picscore.net
```

- 운영 서버

```
https://k12s306.p.ssafy.io
```

- `prometheus` , `cadvisor` 접근 시

```
ID: admin  
PW: pass123#
```

5.7. 추가 개발

5.7.1. 소스 코드 클론

- 프로젝트 소스 코드를 GitLab에서 클론합니다. (로컬 PC)

```
cd ~  
git clone [레포지토리 URL]  
cd [레포지토리 이름]
```

5.7.2. 소스 코드 수정 및 push

- Jenkins와 GitLab에 Webhook이 설정되어 있으므로 소스 코드를 변경 후 `git push` 및 `Merge` 를 하면 자동으로 Jenkins에서 빌드 후 배포가 됩니다.

```
git push
```