포팅 메뉴얼

상세

형상 관리

- `GitLab`

이슈 관리

- `Jira`

커뮤니케이션

- `Mattermost`
- `Notion`

IDE

- `Intellij CE 2023.1.3`
- `Visual Studio Code`

Server

- `AWS EC2 t2.xlarge`
  - `Ubuntu 20.04`
  - `Docker 24.0.6`
  - `Nginx 1.18.0`

Frontend

- `React 18.2.0`
  - `next.js 14.0`
- `node.js 18.16.1`
- `tailwind 3.3.3`
- `redux 6.0.0`

Backend

- `Java OpenJDK 11`
- `SpringBoot`
- `Gradle`
- `Spring Data JPA`
- `Lombok`
- `Hibernate`

Database

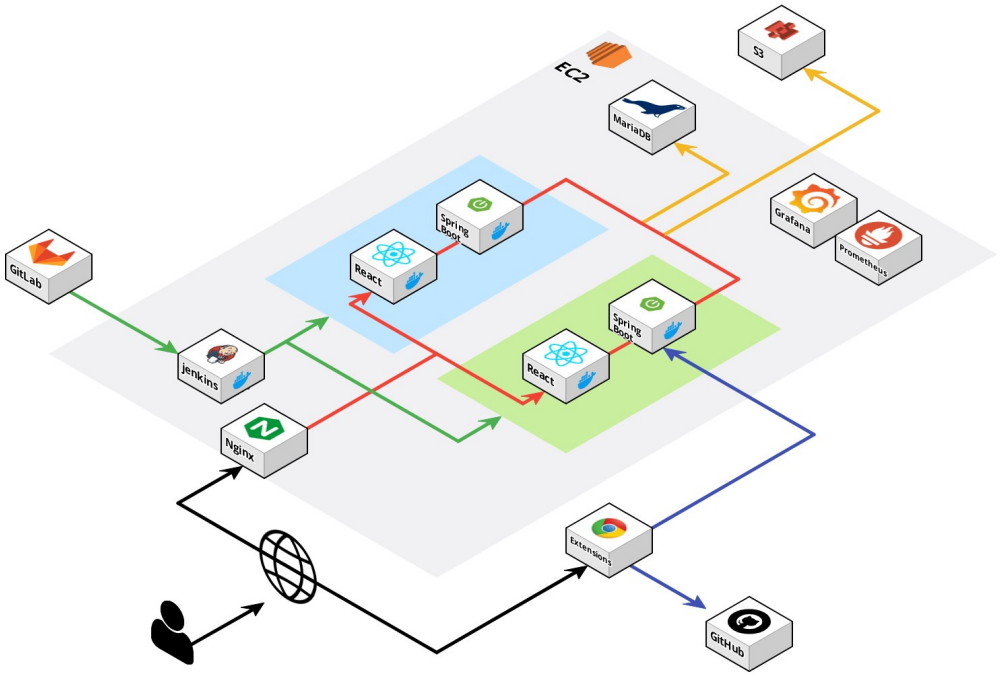
- `MariaDB`

Infra

- `Jenkins 2.414.1`
- `docker-compose 2.16.0`

- `prometheus 2.47.0`
- `grafana 10.1.2`

Infra



SSH 연결 방법(MobaXterm)

```
- MobaXterm을 실행한 뒤 Session → SSH
- Basic SSH settings와 Advanced SSH settings에서 필요한 항목 기입.
  - Remote host에 서버 주소 입력
  - Specify username, 접속할 아이디 입력 (ex: ubuntu)
  - Use private key, PEM 키 파일 등록
```

EC2 인스턴스에 설치된 패키지 목록 업데이트, 업그레이드

```
sudo apt-get update
sudo apt-get upgrade
```

EC2 타임존 한국 표준시로 설정

```
date //현재 타임존 확인
sudo timedatectl set-timezone Asia/Seoul //타임존 한국 표준시로 설정
```

UFW(Ubuntu 방화벽 설정 도구) 설정

```
sudo ufw default deny incoming // 모든 인바운드 연결 차단, 모든 아웃바운드 연결 허용
sudo ufw default allow outgoing
sudo ufw allow ssh // 22번 포트 허용 -> 유의! 허용안하고 방화벽 키면 SSH 접속 안됨.
sudo ufw allow http // 80번 포트 허용
sudo ufw allow https // 443번 포트 허용

sudo ufw enable //방화벽 켜기
```

Nginx 설정

```
sudo apt-get install nginx //nginx 설치
```

```
// etc/nginx/nginx.conf
// 추가 설정은 etc/nginx/conf.d/*.conf 로 만들면 된다.

user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
  worker_connections 768;
  # multi_accept on;
}

http {

  ##
  # Basic Settings
  ##

  sendfile on;
  tcp_nopush on;
  tcp_nodelay on;
  keepalive_timeout 65;
  types_hash_max_size 2048;
  # server_tokens off;

  # server_names_hash_bucket_size 64;
  # server_name_in_redirect off;

  include /etc/nginx/mime.types;
  default_type application/octet-stream;

  upstream backend-service {
      server cogit.kr:8080;
      keepalive 100;
  }

  upstream frontend-service {
      server cogit.kr:3000;
      keepalive 100;
  }

  upstream sonar {
      server 127.0.0.1:9000 fail_timeout=0;
  }

  # HTTP 서버 설정
  server {
```

```
  # 80 포트에서 들어오는 HTTP 요청을 수신
    listen 80;
  # 요청을 처리할 도메인 이름
    server_name cogit.kr www.cogit.kr;
  # 서버 버전 정보 숨기기 (보안상의 이유)
    server_tokens off;
  # 모든 HTTP 요청을 HTTPS로 리다이렉트
    location / {
       return 301 https://$server_name$request_uri;
    }
}

# HTTPS 서버 설정
  server {
  # 443 포트에서 들어오는 HTTPS 요청을 수신
    listen 443 ssl;
    server_name cogit.kr www.cogit.kr;
    server_tokens off;
  # 액세스 로그 기록 비활성화
    access_log off;
  # Let's Encrypt로부터 받은 SSL 인증서와 키 파일 경로
    ssl_certificate /etc/letsencrypt/live/cogit.kr/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/cogit.kr/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf; # SSL 설정 포함
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # DH 파라미터 경로
    keepalive_timeout 1000;

    location / {
       proxy_pass http://frontend-service;
    }
    location /api {
       rewrite ^/api(/.*)$ $1 break;
       proxy_pass http://backend-service;
    }
    location /sonar {
       proxy_pass http://sonar;
    }
}



  ##
  # SSL Settings
  ##

  ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE
  ssl_prefer_server_ciphers on;

  ##
  # Logging Settings
  ##

  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;

  ##
  # Gzip Settings
  ##

  gzip on;

  # gzip_vary on;
  # gzip_proxied any;
  # gzip_comp_level 6;
  # gzip_buffers 16 8k;
  # gzip_http_version 1.1;
  # gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascript

  ##
  # Virtual Host Configs
  ##

  include /etc/nginx/conf.d/*.conf;
  include /etc/nginx/sites-enabled/*;
}


#mail {
# # See sample authentication script at:
# # http://wiki.nginx.org/ImapAuthenticateWithApachePhpScript
#
# # auth_http localhost/auth.php;
# # pop3_capabilities "TOP" "USER";
# # imap_capabilities "IMAP4rev1" "UIDPLUS";
```

```
#
# server {
#   listen     localhost:110;
#   protocol   pop3;
#   proxy      on;
# }
#
# server {
#   listen     localhost:143;
#   protocol   imap;
#   proxy      on;
# }
#}
```

https 적용

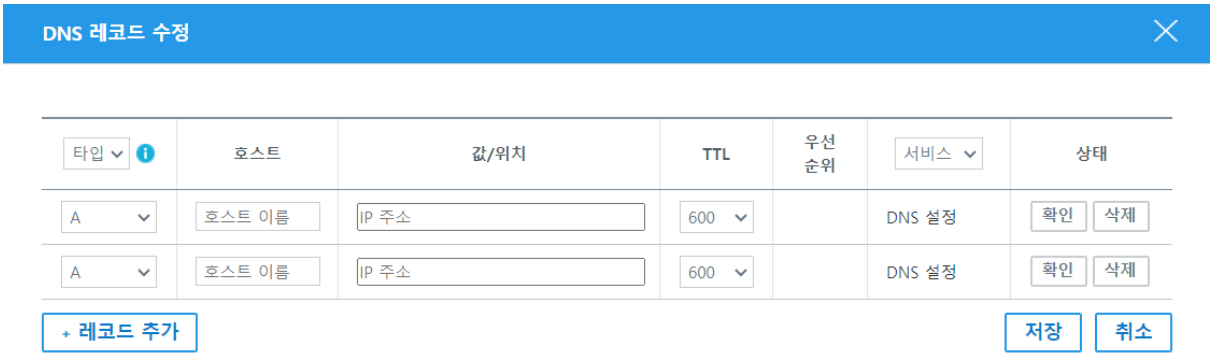
```
sudo snap install core

//기존의 잘못된 certbot 삭제
sudo apt remove certbot

//certbot 설치
sudo snap install --classic certbot

//인증서 받아오기
sudo certbot --nginx
```

DNS 설정




```
curl http://169.254.169.254/latest/meta-data/public-ipv4 //ec2 공인 ip 주소 확인

- 값에 공인 ip 값 넣기
- A 타입에 호스트 이름은 @, www 입력
- TTL은 DNS 레코드의 변경사항이 적용될 때까지 걸리는 시간(초)을 결정하는 DNS 레코드 값
- CNAME은 도메인 주소를 또 다른 도메인 주소로 매핑해준다.
```

Docker 설치

```
# apt 업데이트
sudo apt-get update

# 필수 요소 설치
sudo apt-get install \\
    apt-transport-https \\
    ca-certificates \\
    curl \\
    gnupg-agent \\
```

```
    software-properties-common

# docker gpg 키 설치
curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo apt-key add -
OK

# docker 레포지토리 추가
sudo add-apt-repository \\
   "deb [arch=amd64] <https://download.docker.com/linux/ubuntu> \\
   $(lsb_release -cs) \\
   stable"

# apt 업데이트
sudo apt-get update

# docker 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io

# docker 실행 권한 추가
sudo usermod -aG docker ubuntu

# docker-compose 설치
sudo curl -L <https://github.com/docker/compose/releases/download/v2.16.0/docker-compose-`uname> -s`-`uname -m` -o /usr/local/bin/do
cker-compose

# docker-compose 실행권한 추가
sudo chmod +x /usr/local/bin/docker-compose
```

Backend Dockerfile

```
FROM openjdk:11-jdk

ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar

ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Frontend Dockerfile

```
FROM node:20 as build-stage
WORKDIR /app
COPY package*.json ./
RUN npm install
RUN npm i sharp
COPY . .
RUN npm run build
CMD ["npm", "start"]
```

Jenkins 설치

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
sudo systemctl start jenkins
```

Jenkins 플러그인 설치

- Docker plugin, Docker Pipeline
- GitLab

- Gradle
- NodeJS Plugin
- Publish Over SSH, SSH Agent Plugin, SSH server

Jenkins Pipeline 코드

```
//무중단 배포 전
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS')
    }
    environment {
        CREDENTIAL_ID = 'cogit'
        SOURCE_CODE_URL = credentials('PROJECT_URL')
        RELEASE_BRANCH = 'develop'
        MARIADB_DATABASE_URL = credentials('MARIADB_DATABASE_URL')
        DATABASE_USERNAME = credentials('DATABASE_USERNAME')
        DATABASE_PASSWORD = credentials('DATABASE_PASSWORD')
    GITHUB_CLIENT_ID = credentials('GITHUB_CLIENT_ID')
    GITHUB_CLIENT_SECRET = credentials('GITHUB_CLIENT_SECRET')
    ACCESS_TOKEN_VALID_TIME = credentials('ACCESS_TOKEN_VALID_TIME')
    REFRESH_TOKEN_VALID_TIME = credentials('REFRESH_TOKEN_VALID_TIME')
    JWT_KEY_SIZE_BITS = credentials('JWT_KEY_SIZE_BITS')
    JWT_SECRET_KEY = credentials('JWT_SECRET_KEY')
    S3_BUCKET = credentials('S3_BUCKET')
    S3_ACCESS_KEY = credentials('S3_ACCESS_KEY')
    S3_SECRET_KEY = credentials('S3_SECRET_KEY')
    }
    stages {
        stage('git clone') {
            steps {
                git url: "$SOURCE_CODE_URL",
                    branch: "$RELEASE_BRANCH",
                    credentialsId: "$CREDENTIAL_ID"
                sh "ls -al"
            }
        }
        stage('set backend, frontend environment') {
            steps {
                dir("./Backend/src/main/resources") {
                    sh '''
                    echo "MARIADB_DATABASE_URL: $MARIADB_DATABASE_URL\n\
DATABASE_USERNAME: $DATABASE_USERNAME\n\
DATABASE_PASSWORD: $DATABASE_PASSWORD\n\
GITHUB_CLIENT_ID: $GITHUB_CLIENT_ID\n\
GITHUB_CLIENT_SECRET: $GITHUB_CLIENT_SECRET\n\
ACCESS_TOKEN_VALID_TIME: $ACCESS_TOKEN_VALID_TIME\n\
REFRESH_TOKEN_VALID_TIME: $REFRESH_TOKEN_VALID_TIME\n\
JWT_KEY_SIZE_BITS: $JWT_KEY_SIZE_BITS\n\
JWT_SECRET_KEY: $JWT_SECRET_KEY\n\
S3_BUCKET: $S3_BUCKET\n\
S3_ACCESS_KEY: $S3_ACCESS_KEY\n\
S3_SECRET_KEY: $S3_SECRET_KEY" > "env.yml"
                    '''
                }

                dir("./Backend") {
                    sh "cp ../../config/back/docker-compose.yml ./docker-compose.yml"
                    sh "chmod +x gradlew"
                    sh "./gradlew clean"
                    sh "./gradlew build -x test"
                }
                dir("./Frontend") {
                    sh "cp ../../config/front/docker-compose.yml ./docker-compose.yml"

                }
            }
        }
        stage('down container') {
            steps {
                dir("./Backend") {
                    sh "docker-compose -f docker-compose.yml down --rmi all"
                }
                dir("./Frontend") {
                    sh "docker-compose -f docker-compose.yml down --rmi all"
                }
```

```
            }
        }
        stage('build docker') {
            steps {
                dir("./Backend") {
                    sh "docker-compose -f docker-compose.yml build --no-cache"
                }
                dir("./Frontend") {
                    sh "docker-compose -f docker-compose.yml build --no-cache"
                }
            }
        }
        stage('up container') {
            steps {
                dir("./Backend") {
                    sh "docker-compose -f docker-compose.yml up -d"
                }
                dir("./Frontend") {
                    sh "docker-compose -f docker-compose.yml up -d"
                }
            }
        }
    }
}
```

```
//무중단 배포
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS')
    }
    environment {
        CREDENTIAL_ID = 'cogit'
        SOURCE_CODE_URL = credentials('PROJECT_URL')
        RELEASE_BRANCH = 'develop'
        MARIADB_DATABASE_URL = credentials('MARIADB_DATABASE_URL')
        DATABASE_USERNAME = credentials('DATABASE_USERNAME')
        DATABASE_PASSWORD = credentials('DATABASE_PASSWORD')
    GITHUB_CLIENT_ID = credentials('GITHUB_CLIENT_ID')
    GITHUB_CLIENT_SECRET = credentials('GITHUB_CLIENT_SECRET')
    ACCESS_TOKEN_VALID_TIME = credentials('ACCESS_TOKEN_VALID_TIME')
    REFRESH_TOKEN_VALID_TIME = credentials('REFRESH_TOKEN_VALID_TIME')
    JWT_KEY_SIZE_BITS = credentials('JWT_KEY_SIZE_BITS')
    JWT_SECRET_KEY = credentials('JWT_SECRET_KEY')
    S3_BUCKET = credentials('S3_BUCKET')
    S3_ACCESS_KEY = credentials('S3_ACCESS_KEY')
    S3_SECRET_KEY = credentials('S3_SECRET_KEY')
    PROJECT_URL = credentials('PROJECT_URL')
    DOCKER_APP_NAME = 'backend'

    }
    stages {
        stage('git clone') {
            steps {
                git url: "$SOURCE_CODE_URL",
                    branch: "$RELEASE_BRANCH",
                    credentialsId: "$CREDENTIAL_ID"
                sh "ls -al"
            }
        }
        stage('Remove existing container') {
            steps {
                script {
                    try {
                        sh "docker stop cogit-db"
                        sh "docker rm cogit-db"
                    } catch (Exception e) {
                        echo "Failed to stop and remove cogit-db container. It may not exist, which is fine."
                    }
                }
            }
        }
        stage('set backend, frontend environment') {
            steps {
                dir("./Backend/src/main/resources") {
                    sh '''
                    echo "MARIADB_DATABASE_URL: $MARIADB_DATABASE_URL\n\
DATABASE_USERNAME: $DATABASE_USERNAME\n\
DATABASE_PASSWORD: $DATABASE_PASSWORD\n\
GITHUB_CLIENT_ID: $GITHUB_CLIENT_ID\n\
GITHUB_CLIENT_SECRET: $GITHUB_CLIENT_SECRET\n\
ACCESS_TOKEN_VALID_TIME: $ACCESS_TOKEN_VALID_TIME\n\
```

```
REFRESH_TOKEN_VALID_TIME: $REFRESH_TOKEN_VALID_TIME\n\
JWT_KEY_SIZE_BITS: $JWT_KEY_SIZE_BITS\n\
JWT_SECRET_KEY: $JWT_SECRET_KEY\n\
S3_BUCKET: $S3_BUCKET\n\
S3_ACCESS_KEY: $S3_ACCESS_KEY\n\
S3_SECRET_KEY: $S3_SECRET_KEY\n\
PROJECT_URL: $PROJECT_URL" > "env.yml"
                    '''
                }

        dir("./Backend") {
            // docker compose 파일들과 deploy.sh 가져오기
            sh "sudo cp ../../config/back/docker-compose.blue.yml ./docker-compose.blue.yml"
            sh "sudo cp ../../config/back/docker-compose.green.yml ./docker-compose.green.yml"
            sh "sudo cp ../../config/back/deploy.sh ./deploy.sh"
            // java build
            sh "chmod +x gradlew"
            sh "./gradlew clean"
            sh "./gradlew build -x test"
            // deploy.sh 실행
            sh "chmod +x deploy.sh"
            sh "sudo sh ./deploy.sh"
            }
        dir("./Frontend") {
                sh "sudo cp ../../config/front/.env.development ./.env.development"
                sh "sudo cp ../../config/front/docker-compose.yml ./docker-compose.yml"


        }
            }
        }
        stage('down container') {
            steps {
        dir("./Frontend") {
                    sh "docker-compose -f docker-compose.yml down --rmi all"
                }

            }
        }

        stage('build docker') {
            steps {
                dir("./Frontend") {
                    sh "docker-compose -f docker-compose.yml build --no-cache"
                }
            }
        }
        stage('up container') {
            steps {
                dir("./Frontend") {
                    sh "docker-compose -f docker-compose.yml up -d"
                }
            }
        }

    }
    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                def Build_Duration = currentBuild.durationString
                def Commit_Hash = sh(script: "git rev-parse HEAD", returnStdout: true).trim()

                mattermostSend (
                    color: 'good',
                    message: "빌드 성공\n" +
                             "작성자: ${Author_ID} (${Author_Email})\n" +
                             "커밋: ${Commit_Hash}\n" +
                             "빌드 시간: ${Build_Duration}\n" +
                             "배포 준비 상태: 준비 완료\n" +
                             "[CI 시스템 빌드 링크](http://cogit.kr:8180/job/cogit/)",
                    endpoint: 'https://meeting.ssafy.com/hooks/z4t8cgco4i8uzbstbejekxik9y',
                    channel: 'A109-Monitoring-Alert'
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                def Commit_Hash = sh(script: "git rev-parse HEAD", returnStdout: true).trim()
                def Last_100_Log_Lines = sh(script: "tail -n 100 build.log", returnStdout: true).trim()
```

```
                mattermostSend (
                    color: 'danger',
                    message: "빌드 실패\n" +
                             "작성자: ${Author_ID} (${Author_Email})\n" +
                             "커밋: ${Commit_Hash}\n" +
                             "에러: ${currentBuild.error}\n" +
                             "로그: ```\n${Last_100_Log_Lines}\n```\n" +
                             "[CI 시스템 빌드 링크](http://cogit.kr:8180/job/cogit/)",
                    endpoint: 'https://meeting.ssafy.com/hooks/z4t8cgco4i8uzbstbejekxik9y',
                    channel: 'A109-Monitoring-Alert'
                )
            }
        }
    }
}
```

Backend (blue-green 무중단 배포 방식)

```
//deploy.sh

DOCKER_APP_NAME="backend"

# blue container를 기준으로 docker에 띄워져 있는지 확인
EXIST_BLUE=$(docker-compose -p ${DOCKER_APP_NAME}-blue -f docker-compose.blue.yml ps | grep Up)

# blue container가 띄워져 있지 않다면
if [ -z "$EXIST_BLUE" ]; then
  echo "blue container not activated"
  echo "blue up"

  # blue container 띄우기
  docker-compose -p ${DOCKER_APP_NAME}-blue -f docker-compose.blue.yml up --build -d
  NEW_COLOR="blue"
  OLD_COLOR="green"
else
  echo "blue container activated"
  echo "green up"

  # green container 띄우기
  docker-compose -p ${DOCKER_APP_NAME}-green -f docker-compose.green.yml up --build -d
  NEW_COLOR="green"
  OLD_COLOR="blue"
fi

sleep 10

# 새로 띄운 container가 정상적으로 올라갔는지 확인
EXIST_NEW=$(docker-compose -p ${DOCKER_APP_NAME}-${NEW_COLOR} -f docker-compose.${NEW_COLOR}.yml ps | grep Up)

# 정상적으로 올라갔다면
if [ -n "$EXIST_NEW" ]; then
  # NGINX 설정을 새로운 container에 맞게 변경
  cp /etc/nginx/nginx.${NEW_COLOR}.conf /etc/nginx/nginx.conf
  nginx -s reload

  # 기존에 띄워져 있던 container 내리기
  docker-compose -p ${DOCKER_APP_NAME}-${OLD_COLOR} -f docker-compose.${OLD_COLOR}.yml down
  echo "${OLD_COLOR} container down"
  docker rmi ${DOCKER_APP_NAME}-${OLD_COLOR}-backend
  echo "${DOCKER_APP_NAME}-${OLD_COLOR} image removed"
fi
```

`docker-compose.blue.yml` 과 `docker-compose.green.yml` 파일

```
version: "3.8"
services:
  backend:
    build:
      dockerfile: Dockerfile
    restart: always
```

```
    depends_on:
      - database
    ports:
      - 8080:8080
    container_name: blue
    networks:
      - deploy
    environment:
      - DB_HOST=database

  database:
    image: mariadb:latest
    container_name: cogit-db
    environment:
      MYSQL_ROOT_PASSWORD: cogit109!
      MYSQL_USER: cogit
      MYSQL_PASSWORD: cogit109
      MYSQL_DATABASE: cogit
      MYSQL_CHARACTER_SET_SERVER: utf8mb4
      MYSQL_COLLATION_SERVER: utf8mb4_unicode_ci
      TZ: Asia/Seoul
    ports:
      - 3306:3306
    volumes:
      - cogit-db:/var/lib/mysql
    networks:
      - deploy

networks:
  deploy:
    external: true

volumes:
  cogit-db:
    name: backend_cogit-db
    external: true
```

```
version: "3.8"
services:
  backend:
    build:
      dockerfile: Dockerfile
    restart: always
    depends_on:
      - database
    ports:
      - 8081:8080
    container_name: green
    networks:
      - deploy
    environment:
      - DB_HOST=database

  database:
    image: mariadb:latest
    container_name: cogit-db
    environment:
      MYSQL_ROOT_PASSWORD: cogit109!
      MYSQL_USER: cogit
      MYSQL_PASSWORD: cogit109
      MYSQL_DATABASE: cogit
      MYSQL_CHARACTER_SET_SERVER: utf8mb4
      MYSQL_COLLATION_SERVER: utf8mb4_unicode_ci
      TZ: Asia/Seoul
    ports:
      - 3306:3306
    volumes:
      - cogit-db:/var/lib/mysql
    networks:
      - deploy

networks:
  deploy:
    external: true

volumes:
  cogit-db:
    name: backend_cogit-db
    external: true
```