포팅메뉴얼

사용 도구

- 개발이슈 관리 : Jira
- 형상 관리 : GitLab
- 커뮤니케이션 : Notion, MatterMost
- 디자인 : Figma
- CI/CD : Jenkins

개발 도구

- Visual Studio Code :
- Intellij : 2023.3 (Ultimate Edition)
- Amazon Corretto

개발 기술스택

Frontend

Node.js	18.13.0
React	18.2.0
socket.js	

Backend

Java 17 (Amazon Corretto)
Spring Boot(3.4.1)
MySQL(8.0.40)
JPA
Redis(3.0.504)
OpenVidu(2.31.0)
Spring Security(6.4.2)
OAuth2
WebSocket
STOMP
Lombok(1.18.36)
JWT(0.12.3)

Server

AWS EC2	CPU : Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz RAM : 16GB OS : Ubuntu |
Nginx(1.27.3)	
Jenkins(2.479.3)	
Docker(27.5.1)	
Ubuntu(22.04.4 LTS)	
Prometheus	
Grafana	

환경변수(추후 추가 예정)

```

```

빌드 배포 메뉴얼

1. EC2 인스턴트 준비

1. AWS Ubuntu EC2 인스턴트를 생성
2. 필요한 포트 열기
    - Jenkins: `8080`
    - SSH: `22`
    - 애플리케이션 포트(예: `80`, `443`, `3000` 등).

2. pem키 준비

1. `.pem` 파일을 로컬 컴퓨터에 저장

```
#PowerShell
wsl --install -d Ubuntu
```

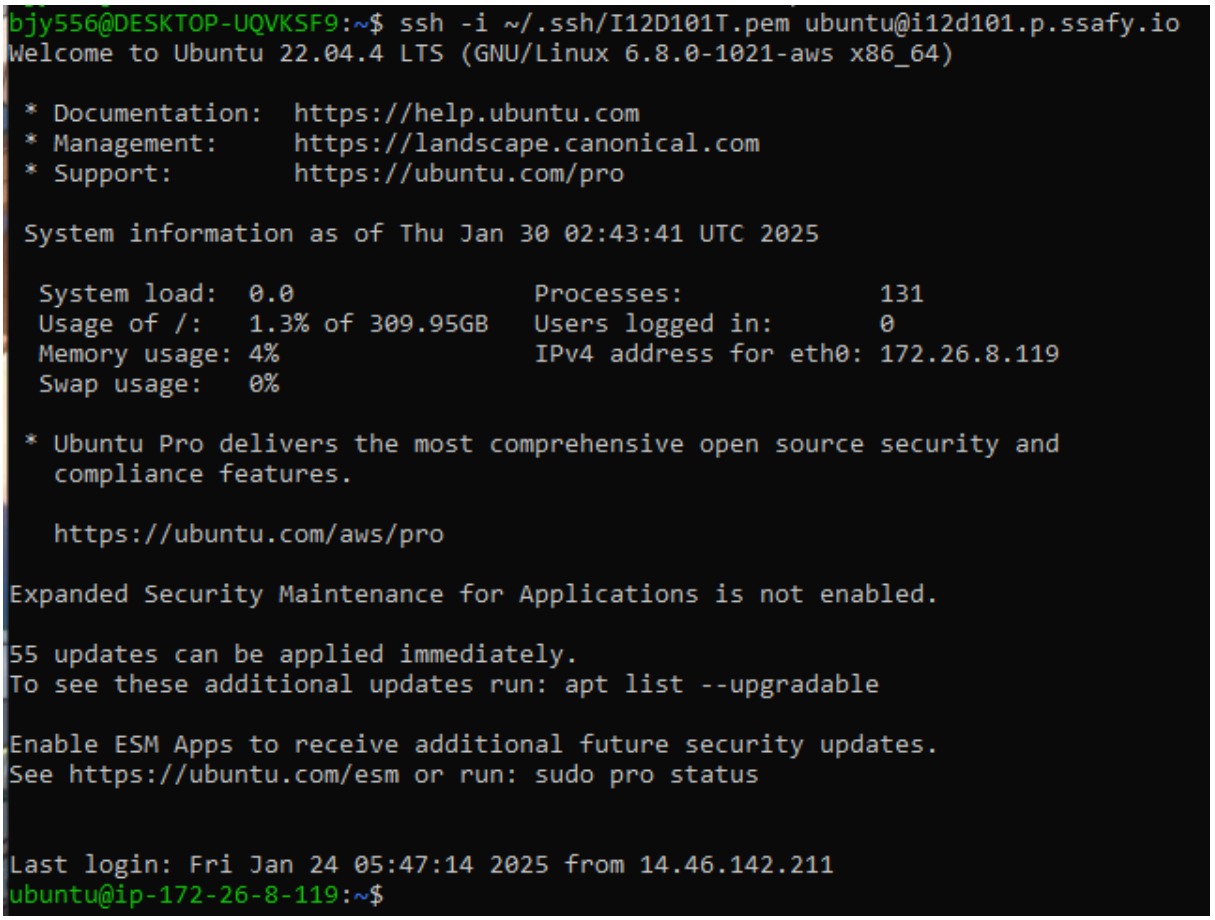
2. ubuntu 계정에 대한 비밀번호 설정

```
sudo passwd
su root
```

3. 서버 접

```
#폴더 생성
mkdir .ssh
# 자신 경로에 pem키 있는지 확인
cd /mnt/c/Users/Bae/Desktop/ssafy/pem
#해당 경로로 복사
cp /mnt/c/Users/Bae/Desktop/ssafy/pem/I12D101T.pem ~/.ssh/
#권한 설정(읽기)
```

```
chmod 400 ~/.ssh/I12D101T.pem
#서버 접속
ssh -i ~/.ssh/I12D101T.pem ubuntu@i12d101.p.ssafy.io
```



3. 우분투 방화벽 설정

```
# 필수 포트 허용
sudo ufw allow 22    # SSH
sudo ufw allow 80    # HTTP
sudo ufw allow 443   # HTTPS
sudo ufw allow 8080  # Jenkins
sudo ufw allow 8081  # Blue Server
sudo ufw allow 8082  # Green Server
sudo ufw allow 3000  # React
sudo ufw allow 3478/tcp  # OpenVidu TURN
sudo ufw allow 3478/udp
```

```
sudo ufw allow 5442/tcp  # OpenVidu Server
sudo ufw allow 5443/tcp
```

4. 도커설치

4.1. Docker 레포지토리 설정

```
# 시스템의 패키지 목록을 최신화
sudo apt-get update

# SSL 인증서와 curl 도구 설치 (보안 통신과 파일 다운로드에 필요)
sudo apt-get install ca-certificates curl

# Docker의 GPG 키를 저장할 디렉토리 생성 (권한: 0755)
sudo install -m 0755 -d /etc/apt/keyrings

# Docker의 공식 GPG 키를 다운로드 (패키지 인증에 사용)
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/key

# 다운로드한 GPG 키를 모든 사용자가 읽을 수 있도록 권한 설정
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

4.2. 레포지토리 추가

```
# Docker 공식 레포지토리를 시스템의 소프트웨어 소스에 추가
# - arch=$(dpkg --print-architecture): 시스템 아키텍처 확인 (예: amd64)
# - VERSION_CODENAME: Ubuntu 버전 코드네임 (예: focal)
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/do
```

4.3. Docker 패키지 설치

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

4.4. Docker 설치 확인 및 권한 설정

```
# 현재 사용자를 docker 그룹에 추가
sudo usermode -aG docker $USER
#변경사항 적용
newgrp docker
#권한 확인
groups
`ubuntu adm dialout cdrom floppy sudo audio dip video plugdev netdev lxd do
```

5. Jenkins 설치

5.1. 호스트 특정 디렉토리에 마운트

```
cd /home/ubuntu && mkdir jenkins-data
```

5.2. Jenkins Docker 컨테이너에 설치

```
docker run -d \
  -v /home/ubuntu/jenkins-data:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /home/ubuntu/docker/proxy:/proxy \
  -p 8080:8080 \
  -e JENKINS_OPTS="--prefix=/jenkins" \
  --group-add $(getent group docker | cut -d: -f3) \
  -e TZ=Asia/Seoul \
  --restart=on-failure \
  --name jenkins \
  jenkins/jenkins:lts-jdk17

  # -v 호스트의 /home/ubuntu/jenkins-data 디렉토리를 컨테이너의 /var/jenkins_ho
  # -v 호스트의 Docker 소켓을 컨테이너에 마운트
  # -v 호스트의 해당 폴더로 마운트
  # -e Jenkins의 URL 접두사를 '/jenkins'로 설정
  # -e Docker 명령어를 젠킨스 내에서 실행할 권한을 부여
  # -- 실패했을 경우 재시작
  # -- jenkins로 이름 지정
  # -- JDK17버전을 명시적으로 지정
```

초기 비밀번호 확인 `docker logs jenkins`

5.3. Jenkins 환경설정

```
cd /home/ubuntu/jenkins-data

mkdir update-center-rootCAs
#Jenkins가 업데이트 센터에 접속할 때 사용할 SSL 인증서를 제공
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-
#Jenkins가 기본 업데이트 센터 대신 Tencent 미러를 사용하도록 설정
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githu
#그 후 재시작
sudo docker restart jenkins
```

5.4. config 보안 설정 확인

```
vi config.xml
#true가 되어 있어야함
<useSecurity>true</useSecurity>
<securityRealm class="hudson.security.HudsonPrivateSecurityRealm">
<disableSignup>true</disableSignup>
```

5.5. Jenkins 초기 설정

1. http://i12d101.p.ssafy.io:8080 에 접속

2. Install suggeted plugins : 초기 플러그인 모두 설치
3. Getting Started - 계정 생성
   계정명 : admin
   암호 : password
   이름 : d101
   이메일주소 user@example.com

4. 접속 주소

5.6. Jenkins내 docker명령어 실행

- DooD 방식

1. **Jenkins 안에 Docker를 설치하기 위해서 Jenkins 컨테이너에 접속**

```
docker exec -it -u root jenkins bash
```

2. **Jenkins 안에 Docker를 설치**

```
# 필요한 패키지 설치
apt-get update
apt-get install -y \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Docker의 공식 GPG 키 추가
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /

# Docker repository 설정
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker
  $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

# 패키지 목록 업데이트
apt-get update

# Docker CLI만 설치
apt-get install -y docker-ce-cli
```

5.7. Credentials 저장

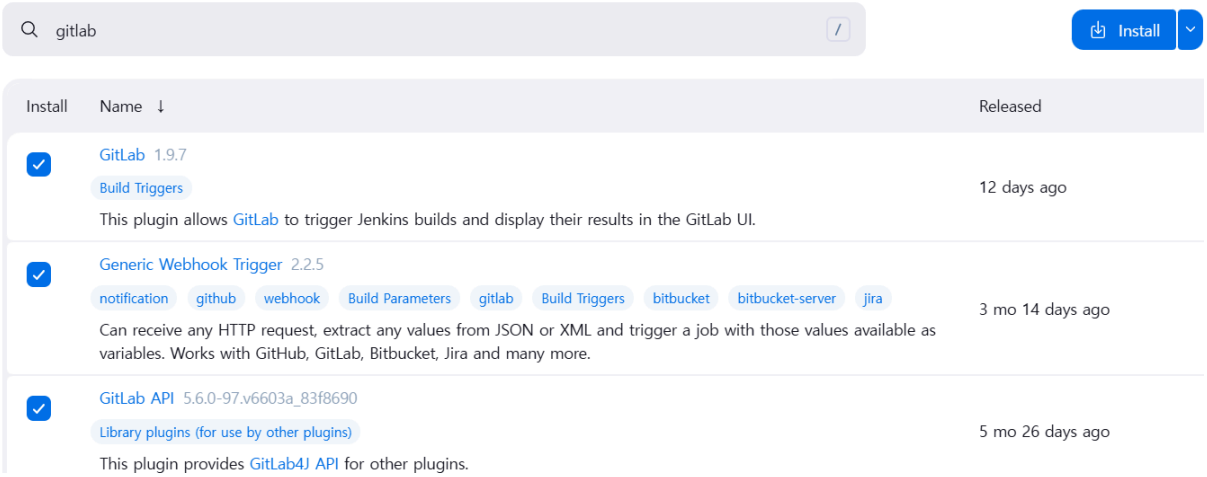
1. .env 작성

```
MYSQL_ROOT_PASSWORD=ssafyD101!
MYSQL_DATABASE=mafia
MYSQL_USER=mafia
MYSQL_PASSWORD=ssafyD101!
```

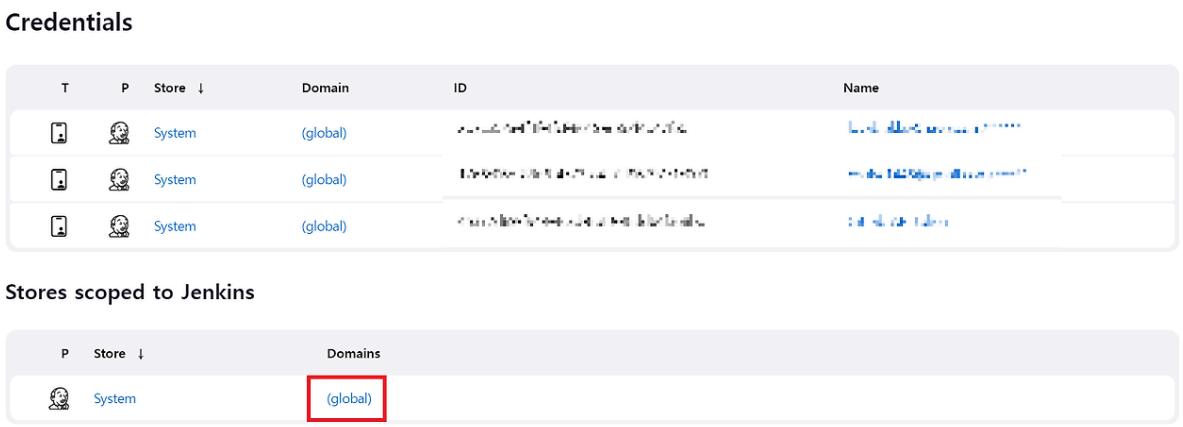
6. Jenkins, Gitlab 연동하기

6.1. Jenkins plugin 설치

Jenkins관리 → Plugins 클릭



6.2. Credential 등록



6.3. Gitlab 연결하기

Gitlab project - Settings - Access Tokens 발급

Access Token

Token name	Scopes	Created	Last Used	Expires	Role	Action
mafia	api, read_api, create_runner, manage_runner, k8s_proxy, read_repository, ai_features	Jan 31, 2025	Never	in 4 weeks	Maintainer	

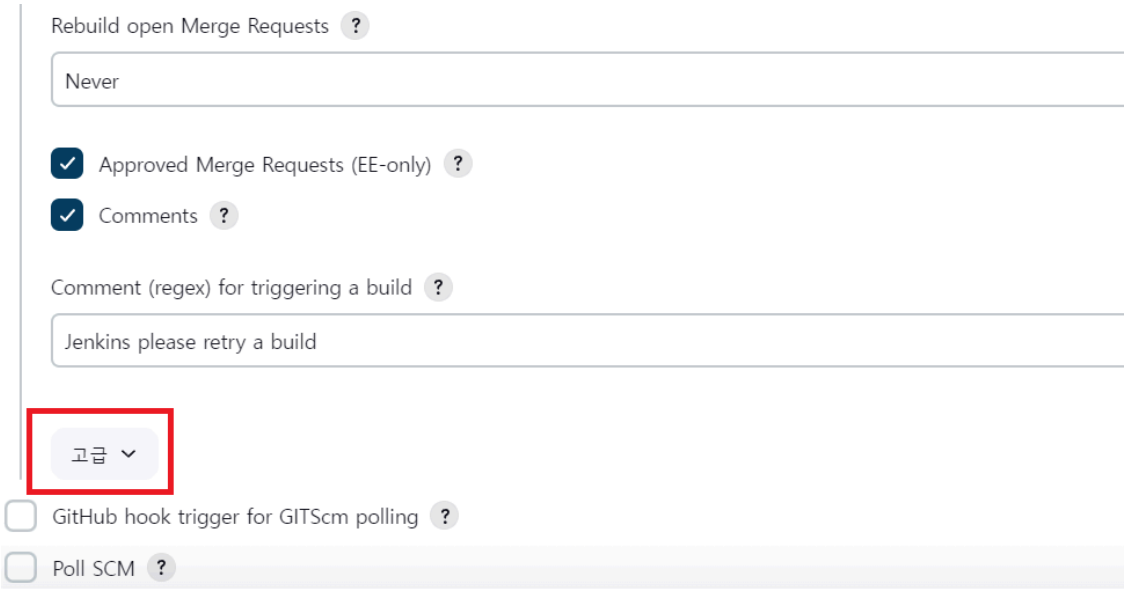
Jenkins 관리 - Credentials - global - add credentials

> user: bjy556@naver.com
> password : Access Token

api token으로 한번 더 credential 만들기

6.4. Pipeline 생성 및 Webhook 연결

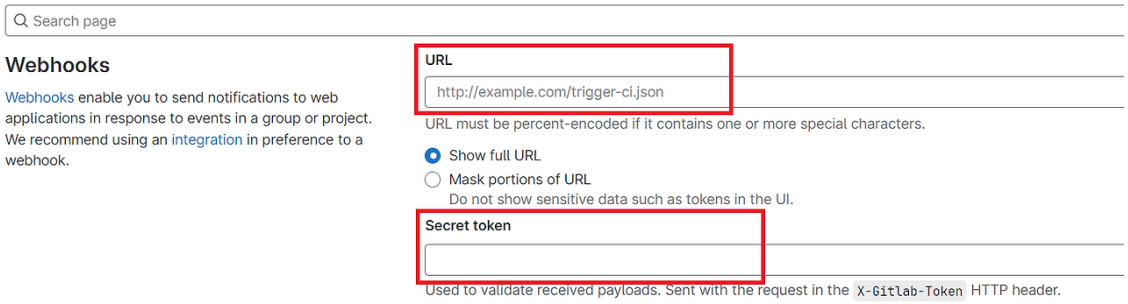
jenkins : 새로운 item → pipeline



- Generate 버튼을 클릭 후
  - jenkins secret token 생성

GitLab : 프로젝트 → setting → webhook

- URL과 jenkins Secret token 입력



7. MySQL, Redis 컨테이너 설치

7.1 docker-compose.yml 작성

```
#home/ubuntu/docker/db/docker-compose.yml
services:
  mysql:
    image: mysql:8
    container_name: mysql
    restart: always
    environment:
```

```
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    ports:
      - "3306:3306"
    volumes:
      - ./mysql/init.sql:/docker-entrypoint-initdb.d/init.sql
      - mysql_data:/var/lib/mysql
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - app-network

  redis:
    image: redis:7
    container_name: redis
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    command: redis-server --requirepass 'Password' --appendonly yes
    networks:
      - app-network

volumes:
  mysql_data:
  redis_data:

networks:
  app-network:
    external: true
```

7.2. .env 설정

```
#docker/db/.env
MYSQL_ROOT_PASSWORD=ssafyD101!
MYSQL_DATABASE=mafia
MYSQL_USER=mafia
MYSQL_PASSWORD=ssafyD101!
```

7.3. docker 명령어 실행

```
#해당 폴더 위치로 이동
cd home/ubuntu/docker/db/docker-compose.yml

docker compose up -d
```

7.4 DB init 설정하기

```
#MYSQL 컨테이너 접속
docker exec -it mysql bash
#MYSQL에 root로 로그인
mysql -u root -p
#password 입력
ssafyD101!
#권한 부여 명령어 실행
USE mafia;
GRANT ALL PRIVILEGES ON mafia.* TO 'mafia'@'%;
FLSUH PRIVILEGES;

#확인하기
SHOW GRANTS FOR 'mafia'@'%';
#나가기
exit;
```

8. NGINX 컨테이너 설치

8.1 docker-compose.yml 작성

```
#home/ubuntu/docker/proxy/docker-compose.yml
services:
   nginx:
     image: nginx:latest
     container_name: nginx
     ports:
       - "80:80"
       - "443:443"
     volumes:
       - ./nginx.conf:/etc/nginx/nginx.conf
       - ./conf.d:/etc/nginx/conf.d
       - /opt/openvidu/certificates:/opt/openvidu/certificates
       - ./data/certbot/www:/var/www/certbot
     command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx -s reload; do
     networks:
       - app-network
     restart: always


#    certbot:
#      image: certbot/certbot
#      volumes:
#        - ./data/certbot/conf:/etc/letsencrypt
#        - ./data/certbot/www:/var/www/certbot
#      entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12
#      depends_on:
#        - nginx

networks:
  app-network:
     external: true
```

8.2 CertBot https 인증서 발급

8.2.1 nginx.conf 작성

```
 upstream backend {
    server blue:8081;
    server green:8082 backup;
}

server {
    listen 80;
    listen [::]:80;
    server_name i12d101.p.ssafy.io;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$server_name$request_uri;
    }
}
```

8.2.2 폴더 생성 및 권한 설정

```
mkdir -p data/certbot/conf
mkdir -p data/certbot/www
sudo chown -R ubuntu:ubuntu data/certbot
sudo chmod -R 755 data/certbot

//인증서 발급 받기
docker compose exec certbot certbot certonly --webroot -w /var/www/certbo
Saving debug log to /var/log/letsencrypt/letsencrypt.log
```

8.2.4 인증서 발급이 성공되면 SSL pem 파일 작성

```
openssl dhparam -out data/certbot/conf/ssl-dhparams.pem 2048
```

8.3 Nginx 작성

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

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

    upstream backend {
        server blue:8081;
        server green:8082 backup;
    }

    upstream frontend {
        server react:80;
    }

    server {
        listen 80;
        listen [::]:80;
        server_name i12d101.p.ssafy.io;

        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
        }

        location / {
            return 301 https://$server_name$request_uri;
        }
    }

    server {
        listen 443 ssl;
```

```
    server_name i12d101.p.ssafy.io;
    server_tokens off;

    ssl_certificate /etc/letsencrypt/live/i12d101.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i12d101.p.ssafy.io/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass http://frontend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api/ {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    # OAuth2 경로 추가
    location /oauth2/ {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    # login 리다이렉트 경로 추가
    location /login/oauth2/code/ {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /ws-mafia {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
    }

    location /jenkins {
        proxy_pass http://jenkins:8080/jenkins;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Jenkins 관련 추가 설정
        proxy_set_header X-Jenkins-Context "/jenkins";
        proxy_redirect http:// https://;
    }

    add_header X-Content-Type-Options "nosniff" always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-XSS-Protection "1; mode=block" always;
```

```
        add_header Referrer-Policy "no-referrer" always;
        add_header Permissions-Policy "geolocation=()" always;
    }
}
```

9. BackEnd (BLUE,GREEN) 배포

9.1 app/docker-compose.yml 작성

```
#infra/docker/app/docker-compose.yml
services:
  blue:
    build:
      context: ../../../backend/mafia
      dockerfile: Dockerfile
      args:
        - PROFILE=blue
    image: blue
    container_name: blue
    ports:
      - "8081:8081"
    environment:
      - SPRING_PROFILES_ACTIVE=blue
      - PROFILE=blue
      - SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/${MYSQL_DATAB
      - SPRING_DATASOURCE_USERNAME=${MYSQL_USER}
      - SPRING_DATASOURCE_PASSWORD=${MYSQL_PASSWORD}
      - SERVER_PORT=8081
    restart: always
    networks:
      - app-network

  green:
    build:
      context: ../../../backend/mafia
      dockerfile: Dockerfile
```

```
      args:
        - PROFILE=green
    image: green
    container_name: green
    ports:
      - "8082:8082"
    environment:
      - SPRING_PROFILES_ACTIVE=green
      - PROFILE=green
      - SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/${MYSQL_DATAB
      - SPRING_DATASOURCE_USERNAME=${MYSQL_USER}
      - SPRING_DATASOURCE_PASSWORD=${MYSQL_PASSWORD}
      - SERVER_PORT=8082
    restart: always
    networks:
      - app-network

networks:
  app-network:
    external: true
```

9.2. DockerFile 작성

```
#backend/mafia/Dockerfile
FROM amazoncorretto:17
ARG JAR_FILE=./build/libs/mafia-0.0.1-SNAPSHOT.jar
ARG PROFILE
ENV SPRING_PROFILES_ACTIVE=${PROFILE}
WORKDIR /app
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-Dspring.profiles.active=${SPRING_PROFILES_ACTIVE}
```

9.3. Jenkins pipeline을 이용하여 무중단 배포

```
pipeline {
    agent any
    triggers {
```

```
        gitlab(
            triggerOnPush: false,
            triggerOnMergeRequest: false,
            triggerOnAcceptedMergeRequest: true,
            branchFilterType: 'NameBasedFilter',
            includeBranchesSpec: 'dev_BE'
            )
    }
    stages {
        stage('Check Branch') {
            steps {
                script {
                    def targetBranch = env.gitlabTargetBranch ?: env.BRANCH_NAME
                    if (targetBranch != 'dev_BE') {
                        currentBuild.result = 'ABORTED'
                        error("This pipeline only runs for merge requests to dev_BE bra
                    }
                }
            }
        }
        stage('Clone') {
            steps {
                echo 'Start cloning mafia project...'
                git branch: 'dev_BE',
                    credentialsId: 'junyoung',
                    url: 'https://lab.ssafy.com/s12-webmobile1-sub1/S12P11D101.git'
                    echo 'Clone finished!'
            }
        }
        stage('Build') {
            steps {
                echo 'Start building mafia project...'
                dir('backend/mafia') {
                    withCredentials([file(credentialsId: 'application-secret', variable: 'S
                        sh """
                            cat "\$SECRET_FILE" > src/main/resources/application-secre
                            cat src/main/resources/application-secret.yml
                        """
```

```
        }
            sh '''
                chmod +x ./gradlew
                ./gradlew clean build -x test
            '''
        }
        echo 'Build finished!'
    }
}

stage('Check Current Environment') {
    steps {
        script {
            def nginxConfig = sh(script: 'cat /proxy/conf.d/upstream.conf', ret
            CURRENT_BACKEND = nginxConfig.contains('server blue:8081;')
            TARGET_BACKEND = CURRENT_BACKEND == 'blue' ? 'green' : 'b
            echo "Current backend: ${CURRENT_BACKEND}"
            echo "Target backend: ${TARGET_BACKEND}"
        }
    }
}



stage('Deploy New Environment') {
    steps {
        script {
            dir('infra/docker/app') {
                sh 'chmod u+w .'
                withCredentials([file(credentialsId: 'db-credentials', variable: 'DI
                    sh """
                        cat \$DB_FILE > .env
                        chmod 600 .env
                    """
                }

                // 새로운 환경만 배포
                sh """
```

```
                    docker compose build --no-cache ${TARGET_BACKEND}
                    docker compose up -d ${TARGET_BACKEND}
                """

                sh "sleep 20"
            }
        }
    }
    stage('Switch Traffic') {
        steps {
            script {
                // upstream 설정 업데이트
                def newConfig = """
upstream backend {
   server ${TARGET_BACKEND}:${TARGET_BACKEND == 'blue' ? '8081' : '808
    }"""
                sh """
                    echo '${newConfig}' > /proxy/conf.d/upstream.conf
                    docker exec nginx nginx -s reload
                """
            }
        }
}

     stage('Cleanup Old Environment') {
        steps {
            script {
                // 이전 환경 정리 (선택적)
                sleep 30 // 기존 연결이 종료될 때까지 대기
                dir('infra/docker/app') {
                    sh """
                        docker compose stop ${CURRENT_BACKEND}
                        docker compose rm -f ${CURRENT_BACKEND}
                    """
                }
            }
        }
```

```
        }


        stage('Nginx Reload') {
            steps {
                sh '''
                    echo "Reloading Nginx configuration..."
                    docker exec -t nginx nginx -s reload
                    echo "Nginx reload completed!"
                '''
            }
        }

    }
    post {
        success {
            echo 'Pipeline succeeded!'
        }
        failure {
            echo 'Pipeline failed!'
            dir('infra') {
                sh 'docker compose logs'
            }
        }
    }
}
```

10. FrontEnd 배포

10.1. frontend/docker-compose.yml 작성

```
#infra/docker/frontend/docker-compose.yml
services:
  react:
    build:
      context: ../../../frontend/mafia
      dockerfile: Dockerfile
    container_name: react
```

```
    ports:
      - "3000:80"
    networks:
      - app-network
    restart: always

networks:
  app-network:
    external: true
```

10.2. DockerFile 작성

```
# frontend/mafia/Dockerfile
# 빌드 단계
FROM node:22.13.0-alpine as builder

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .
RUN npm run build

# 프로덕션 단계
FROM nginx:alpine

# Nginx 설정 파일 복사 (필요한 경우)
COPY nginx.conf /etc/nginx/conf.d/default.conf

# 빌드된 파일을 Nginx 서버로 복사
COPY --from=builder /app/dist /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

10.3. Nginx.conf 작성

```
# frontend/mafia/nginx.conf
server {
    listen 80;
    location / {
        root /usr/share/nginx/html;
        try_files $uri $uri/ /index.html;
    }
}
```

10.4. Jenkins pipeline를 이용하여배포

```
pipeline {
    agent any

    triggers {
        gitlab(
            triggerOnPush: true,
            triggerOnMergeRequest: false,
            triggerOnAcceptedMergeRequest: true,
            branchFilterType: 'NameBasedFilter',
            includeBranchesSpec: 'dev_FE'
        )
    }

    stages {
        stage('FE-Check Branch') {
            steps {
                script {
                    def targetBranch = env.gitlabTargetBranch ?: env.BRANCH_NAME
                    if (targetBranch != 'dev_FE') {
                        currentBuild.result = 'ABORTED'
                        error("This pipeline only runs for merge requests to dev_FE brar
                    }
                }
            }
        }
```

```
stage('Clone') {
    steps {
        echo 'Start cloning frontend project...'
        git branch: 'dev_FE',
            credentialsId: 'junyoung',
            url: 'https://lab.ssafy.com/s12-webmobile1-sub1/S12P11D101.git'
        echo 'Clone finished!'
    }
}

stage('FE-Build') {
    steps {
        echo 'Start building frontend project...'
        nodejs(nodeJSInstallationName: 'NodeJS 22.13.0') {
            dir('frontend/mafia') {
                sh '''
                    npm install
                    npm run build
                '''
            }
        }
        echo 'Build finished!'
    }
}

stage('FE-Deploy') {
    steps {
        script {
            dir('infra/docker/frontend') {
                sh """
                    docker compose down || true
                    docker compose build --no-cache
                    docker compose up -d
                """
                sh """
                        echo "Reloading Nginx configuration..."
                        docker exec -t nginx nginx -s reload
```

```
                        echo "Nginx reload completed!"
                    """
                // 컨테이너 상태 확인
                sh '''
                    echo "Waiting for containers to be healthy..."
                    sleep 30
                    docker ps
                '''
            }
        }
    }
}

post {
    success {
        echo 'Frontend pipeline succeeded!'
    }
    failure {
        echo 'Frontend pipeline failed!'
        dir('infra/docker/frontend') {
            sh 'docker compose logs'
        }
    }
}
```

11. OpenVidu

11.1. OpenVidu 설치

```
# root 권한 얻기
sudo su
# openvidu 설치 권장 경로 /opt로 이동
cd /opt
```

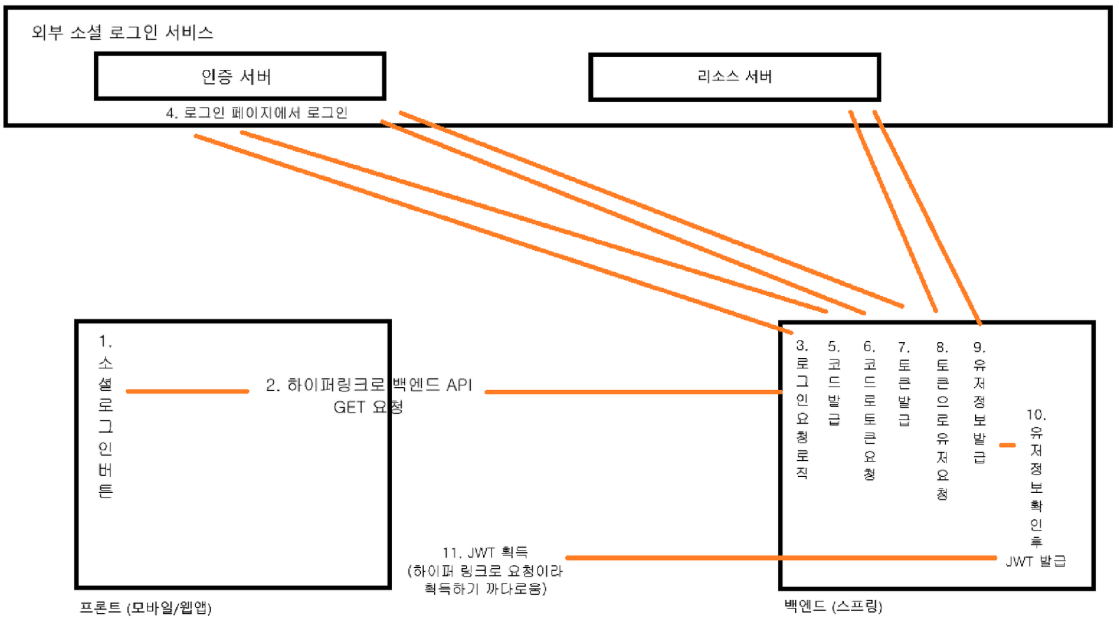
```
# openvidu 설치
curl <https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu
# 설치 후 오픈비두가 설치된 경로로 이동
cd openvidu
# 환경 설정
nano .env
# ----------------------
# OpenVidu Configuration
# 도메인 또는 퍼블릭IP 주소
DOMAIN_OR_PUBLIC_IP=i12d101.p.ssafy.io
# 오픈비두 서버와 통신을 위한 시크릿
OPENVIDU_SECRET=junyoung
# Certificate type
CERTIFICATE_TYPE=letsencrypt
# 인증서 타입이 letsencrypt일 경우 이메일 설정
LETSENCRYPT_EMAIL=user@example.com
# HTTP port
# 최초 1회 실행 후 바꿔야합니다!!(certbot 인증서 발급용)
HTTP_PORT=8442
# HTTPS port(해당 포트를 통해 오픈비두 서버와 연결)
# 최초 1회 실행 후 바꿔야합니다!!(certbot 인증서 발급용)
HTTPS_PORT=8443
```

11.2. Openvidu 실행

```
#openvidu 실행
./openvidu start
-----------------------------------------------------

  OpenVidu Platform is ready!
  ---------------------------

  * OpenVidu Server: https://DOMAIN_OR_PUBLIC_IP/

  * OpenVidu Dashboard: https://DOMAIN_OR_PUBLIC_IP/dashboard/

-----------------------------------------------------
```

KAKAO Outh

소셜 로그인 인증 프로세스 흐름도



1. 프론트단에서 백엔드의 OAuth2 로그인 경로로 하이퍼링킹
2. 진행 후 백엔드단에서 (로그인 페이지 요청 → 코드 발급 → Access 토큰 → 유저 정보 획득 → JWT 발급)

카카오에 APP추가하기

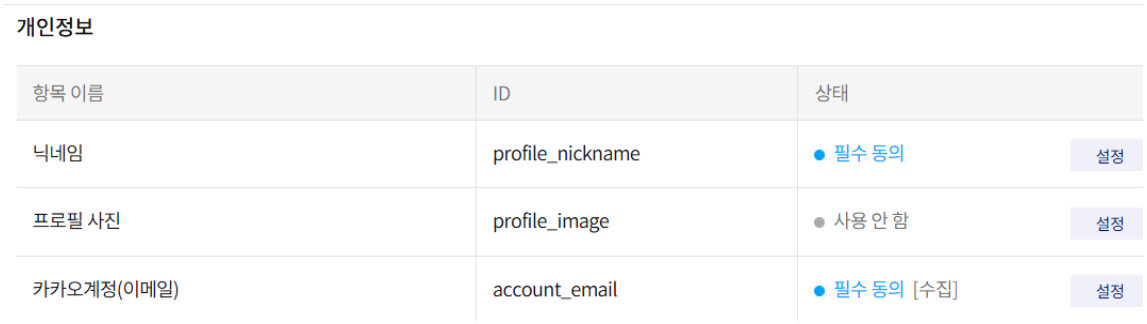
- 카카오 개발자 페이지 들어가서 내 어플리케이션 추가
- 대시보드 - 설정 - 카카오로그인

1. 카카오 로그인 ON

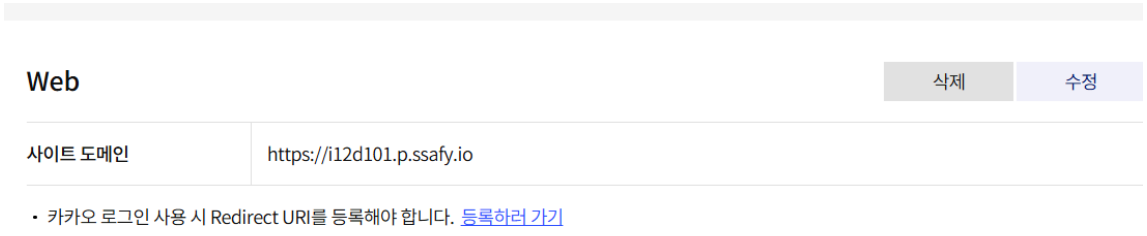
2. RedirectUrl 설정

```
https://i12d101.p.ssafy.io/login/oauth2/code/kakao
```

3. 동의 항목 설정



- 내 어플리케이션 → 앱 설정 → 플랫폼에서 사이트 도메인 등록



application-secret.yml에 추가하기

```
security:
  oauth2:
    client:
```

```
    registration:
      kakao:
        client-id: 611f4a4da2367fac37c65cec2e32a0ee
        client-secret: ubsvxwALetb9oNc6OZnVePQOU9NbAwcM
        redirect-uri: ${app.baseUrl}/login/oauth2/code/kakao
        authorization-grant-type: authorization_code
        client-authentication-method: client_secret_post
        scope: profile_nickname,account_email

    provider:
      kakao:
        authorization-uri: https://kauth.kakao.com/oauth/authorize
        token-uri: https://kauth.kakao.com/oauth/token
        user-info-uri: https://kapi.kakao.com/v2/user/me
        user-name-attribute: id
```