포팅메뉴얼

목차

1. 기술 스택 버전

Front-End

- React.js 18.3.1
- CSS
- JavaScript
- HTML5

Back-End

- Java 17
- SpringBoot 3.3.1

DB

- MySQL 8.0.23
- Redis

- AWS S3

Infra

- Ubuntu 20.04
- Docker
- Jenkins
- AWS EC2
- Nginx

AI

- Gemini
- Karlo

Web RTC / socket

- Openvidu 2.29.0

2. CI / CD - Jenkins

EC2 접속

1. Putty 설정 및 pem키 Puttygen 사용하여 ppk로 변경
2. ssh 22번으로 연결 키 등록해서
3. 타임존 설정

```
sudo rm /etc/localtime
sudo ln -s /usr/share/zoneinfo/Asia/Seoul /etc/localtime
```

4. HostName 변경

   여러 서버를 관리 중일 경우 IP만으로 어떤 서비스의 서버인지 확인하기 어렵기 때문에 설정

```
sudo hostnamectl set-hostname safefoodtruck

// hostname 확인
hostname
```

5. 인스턴스에 있는 모든 패키지 업데이트

```
// window
sudo apt-get update

// mac
sudo yum update -y
```

5. docker 설치

```
// https사용하기 위한 패키지 다운로드 및 설치
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties

// GPG키 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

// Docker의 공식 apt 저장소를 추가
```

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_rele

// 시스템 패키지 업데이트
sudo apt-get update

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

// 도커 설치 버전 확인
docker -v

// 도커 실행상태 확인
sudo systemctl status docker

// 도커 실행
sudo docker run hello-world
```

6. Jenkins Docker 설치

```
sudo docker run --name jenkins
-p 8080:8080
-p 50000:50000
-d
-v jenkins_home:/var/jenkins_home
-v /var/run/docker.sock:/var/run/docker.sock
-v /usr/bin/docker:/usr/bin/docker
jenkins/jenkins:lts-jdk17
```

3. Jenkins 접속
4. Pipeline 생성
5. Gitlab webhook 설정
6. Jenkins + Gitlab web hook 연결 및 트리거 설정, 자격증명 설정
7. Jenkins tool 에서 nodejs, gradle 추가 설정
8. Pipeline Code 작성

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리에서 'master' 브랜치를 체크아웃
                git branch: 'master',
                url: 'https://lab.ssafy.com/s11-webmobile1-sub2/S11P12B102.git',
                credentialsId: 'gitlabC'
            }
        }
    }
}
```

9. Gitlab 연결 상태 및 Stage 정상 실행 테스트
10. CI / CD Pipeline 작성

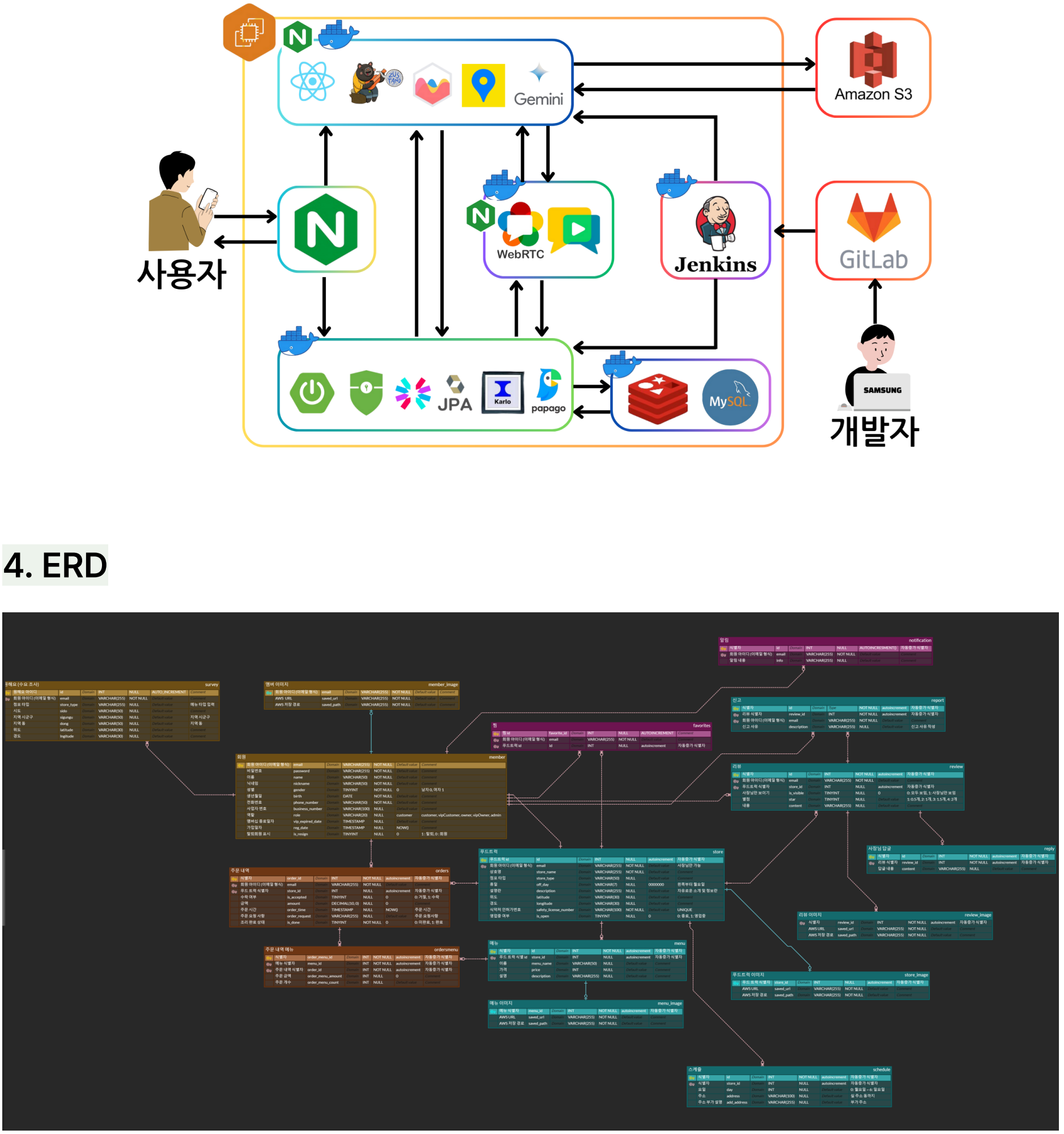
```
pipeline {
    agent any
```

```
    environment {
        SPRING_IMAGE = 'spring-app'
        COMPOSE_FILE = 'docker-compose.yml'
    }
    tools {
        nodejs 'nodejs'
        gradle 'gradle'
    }

    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리에서 'deploy' 브랜치를 체크아웃 //
                git branch: 'deploy',
                url: 'https://lab.ssafy.com/s11-webmobile1-sub2/S11P12B102.git',
                credentialsId: 'gitlabC'
            }
        }

        stage('Build Spring Backend') {
            steps {
                script {
                    dir('Back-End') {
                        sh 'chmod +x gradlew'
                        sh './gradlew clean'
                        sh './gradlew build'
                    }
                }
            }
        }

        stage('Build React Frontend') {
            steps {
                script {
                    dir('Front-End') {
                        sh 'npm install'
                        sh 'CI=false npm run build'
                    }
                }
            }
        }

        stage('Stop and Remove Existing Containers') {
            steps {
                script {
                    // 기존 컨테이너 중지 및 제거
                    sh 'docker-compose -f ${COMPOSE_FILE} down'
                }
            }
        }

        stage('Remove Existing Docker Images') {
            steps {
                script {
                    // 기존 Spring 및 React 이미지를 삭제
                    sh "docker rmi -f \$(docker images -q ${SPRING_IMAGE}) || true"
                }
            }
```

```
        }

        stage('Build Docker Image Spring') {
            steps {
                script {
                    dir('Back-End'){
                        sh 'docker build -t ${SPRING_IMAGE} .'
                    }
                }
            }
        }

        stage('docker-compose up') {
            steps {
                script {
                    sh 'docker-compose -f ${COMPOSE_FILE} down'  // 기존 컨테이너 중지 및 제거
                    sh 'docker-compose -f ${COMPOSE_FILE} up -d' // 새로운 컨테이너 시작
                }
            }
        }

        stage('Copy Built Files to Nginx Directory') {
            steps {
                script {
                    // 빌드된 React 파일을 Nginx 컨테이너의 경로로 복사
                    sh 'docker cp Front-End/build/. nginx-react:/usr/share/nginx/html/'
                }
            }
        }

        stage('Nginx Restart') {
            steps {
                script {
                    sh 'docker restart nginx'
                    sh 'docker restart nginx-react'
                }
            }
        }
    }
}
```

3. Architecture

4. ERD



5. Java

application.yml

```
server:
  servlet:
    context-path: /api

spring:
  profiles:
    include: key

  datasource:
```

```
    url: jdbc:mysql://i11b102.p.ssafy.io:${dbport}/safefoodtruck
    username: root
    password: ${mysqlpassword}
    driver-class-name: com.mysql.cj.jdbc.Driver
    hikari:
      maximum-pool-size: 10

  mail:
    host: [smtp.gmail.com](http://smtp.gmail.com/)
    port: 587
    username: ${google-email}
    password: ${google-app-password}
    properties:
      mail.smtp.debug: true
      mail.smtp.connectiontimeout: 1000
      mail.starttls.enable: true
      mail.smtp.auth: true

  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        diarect: org.hibernate.dialect.MysqlDialect
        jdbc:
          batch_size: 1000
        show_sql: false
        format_sql: false
        use_sql_comments: false
        order_updates: true
        default_batch_fetch_size: 16
    open-in-view: false

  data:
    redis:
      host: [i11b102.p.ssafy.io](http://i11b102.p.ssafy.io/)
      port: 6379
      password: ${redispassword}

springdoc:
  swagger-ui:
    path: ${swagger-url}

logging:
  level:
    org.hibernate.SQL: info

jwt:
  expiration_time: 3600000 #1시간
  secret: ${JWT}

kakao:
  grant-type: authorization_code
  client-id: ${kakao-client-id}
  redirect-uri: ${kakao-redirect-uri

openvidu:
```

```
  url: ${OpenviduURL}
  secret: ${OpenviduSecret}
```

6. Nginx

1. Docker 설치

```
sudo apt-get update
sudo apt-get install docker.io
```

```
sudo systemctl start docker
sudo systemctl enable docker  # 부팅 시 Docker가 자동으로 시작되게 설정
```

2. Nginx 이미지 다운로드

```
sudo docker pull nginx
```

3. Nginx 컨테이너 실행

```
sudo docker run -d --name nginx \
    -p 80:80 \
    -p 443:443 \
  -v ./templates:/etc/nginx/templates \
  -v /home/ubuntu/templates/logs:/var/log/nginx \
    -v 호스트 키 위치:Nginx 키 위치:ro#읽기전용 \
  nginx
```

https를 위한 키 위치 마운트

볼륨 연결 및 로그 파일 위치 설정

4. Nginx Config 설정

```
server {
    listen 443 ssl;
    server_name i11b102.p.ssafy.io;

    ssl_certificate /ssh key 경로/fullchain.pem;
    ssl_certificate_key /ssh key 경로/privkey.pem;

    location / {
        proxy_pass http://프론트 컨테이너 이름:3000; # 프론트엔드 로드 밸런서로 요청 전달
        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://스프링 컨테이너 이름:8080; # 백엔드 로드 밸런서로 요청 전달
        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 /SSE 설정 uri/ {
        proxy_pass http://스프링 컨테이너 이름:8080;
        SSE 설정 추가
    }
}

server {
    listen 80;
    server_name i11b102.p.ssafy.io;

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

7. Docker

Docker Container

```
docker ps

IMAGE                                NAMES                       PORTS
nginx                                nginx-react                 80/tcp
spring-app                           spring                        -
redis                                redis                       0.0.0.0:6379->6379/tcp, :::637
openvidu/openvidu-server:2.30.0      openvidu-openvidu-server-1    -
kurento/kurento-media-server:7.0.1   openvidu-kms-1                -
openvidu/openvidu-coturn:2.30.0      openvidu-coturn-1           0.0.0.0:3478->3478/tcp, 0.0.0
openvidu/openvidu-proxy:2.30.0       openvidu-nginx-1              -
openvidu/openvidu-call:2.30.0        openvidu-app-1                -
nginx                                nginx                       0.0.0.0:80->80/tcp, :::80->80/
ngrinder/agent                       agent
ngrinder/controller                  controller                  0.0.0.0:12000-12099->12000-120
mysql:8.0.37                         mysql                       0.0.0.0:3306->3306/tcp, :::330
jenkins/jenkins:lts-jdk17            jenkins                     0.0.0.0:8080->8080/tcp, :::808
```

8. 외부 서비스 가입 및 활용에 필요한 정보

1. AWS S3

`aws-key-id`

`bucket-name`

`AWS-region`

`access key`

2. Kakao Map

`App Key`

3. Kakao Social Login

`kakao-cliend-id`

`kakao-redirect-uri`

4. Google Login

`google-client-id`

5. Google SMTP

`google-email`

`google-app-password`

6. Openvidu

`openvidu-secret`

`openviduURL`

7. Karlo

`App Key`

8. Papago

`papago-cliend-id`

`papago-cliend-secret`

9. Gemini

`App Key`

10. 사업자등록정보 조회

`Api Key`

11. 푸드트럭지정현황 조회

`Api Key`

9. MySQL

```
docker run --name mysql
-e MYSQL_ROOT_PASSWORD=`사용자 지정 비밀번호 입력 칸`
-d
-p 3306:3306
-v /home/ubuntu/mysql_data:/var/lib/mysql
mysql:8.0.37
```

10. Redis

Redis Docker를 활용하여 사용

```
docker run -it --network some-network --rm redis redis-cli -h some-redis
```

11. SpringBoot Build

Jenkins Pipeline에서 빌드

```
        stage('Build Spring Backend') {
            steps {
                script {
                    dir('Back-End') {
```

```
                        sh 'chmod +x gradlew'
                        sh './gradlew clean'
                        sh './gradlew build -x test'
                    }
                }
            }
        }
```

빌드 후 dockerfile에서 이미지 빌드

```
FROM openjdk:17-alpine
ARG JAR_FILE=build/libs/sft-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

이미지 빌드 후 docker-compose에서 이미지

```
services:
  spring:
    container_name: spring
    image: spring-app
    networks:
      - app-network

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

12. React Build

Jenkins container 위에서 빌드

```
        stage('Build React Frontend') {
            steps {
                script {
                    dir('Front-End') {
                        sh 'npm install'
                        sh 'CI=false npm run build'
                    }
                }
            }
        }
```

빌드한 파일을 React를 실행시킬 Nginx container로 복사

```
        stage('Copy Built Files to Nginx Directory') {
            steps {
                script {
                    // 빌드된 React 파일을 Nginx 컨테이너의 경로로 복사
                    sh 'docker cp Front-End/build/. nginx-react:/usr/share/nginx/html/'
                }
```

```
            }
        }
```

React용 Nginx에서 경로 연결

```
server {
    listen 3000;
    server_name nginx-react;

    location / {
        root /usr/share/nginx/html;
    }
}
```

13. 방화벽 설정 및 Port 관리 (외부)

```
sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22                         ALLOW       Anywhere
443                        ALLOW       Anywhere
7070                       ALLOW       Anywhere      // nGrinder
8443                       ALLOW       Anywhere      // Openvidu
80                         ALLOW       Anywhere
8080                       ALLOW       Anywhere      // Jenkins
22 (v6)                    ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)
7070 (v6)                  ALLOW       Anywhere (v6) // nGrinder
8443 (v6)                  ALLOW       Anywhere (v6) // Openvidu
80 (v6)                    ALLOW       Anywhere (v6)
8080 (v6)                  ALLOW       Anywhere (v6) // Jenkins
```