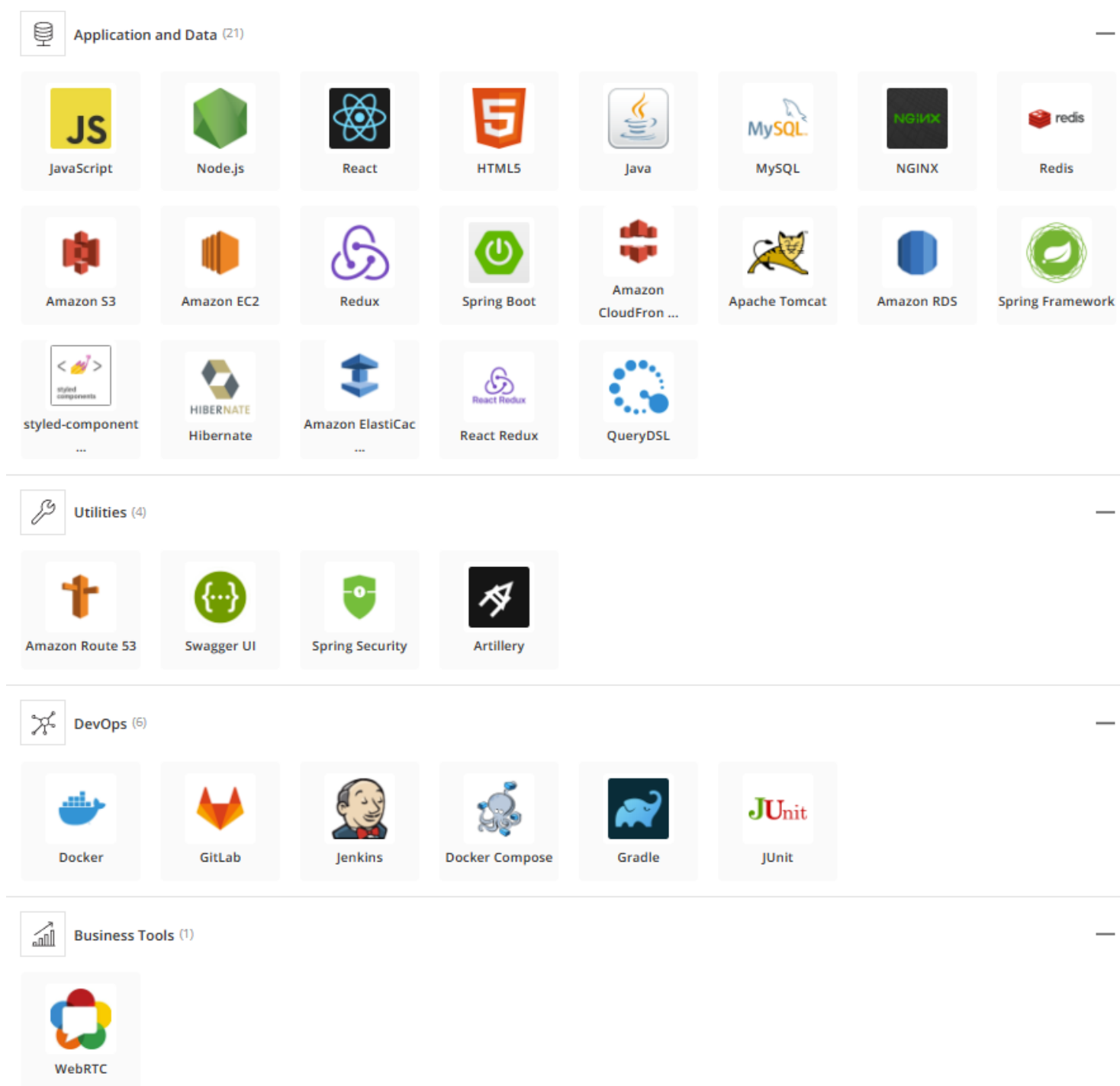


## 포팅 메뉴얼

1. 프로젝트 구성도  
기술 스택  
시스템 아키텍처
2. Spring 서버 ec2 세팅  
docker & docker-compose 설치
3. nginx 서버 ec2 세팅  
docker & docker-compose 설치
4. jenkins 서버 ec2 세팅  
필요 플러그인 설치  
Publish Over SSH 설정  
Mattermost notification 설정  
파이프라인 생성  
jenkins gitlab 웹훅 설정
5. nginx ec2 설정
6. Spring 서버 배포 웹스크립트 설정
7. nginx 배포 설정
8. 프론트엔드 리액트 프로젝트 배포
9. Openvidu 서버 배포 설정
10. RDS 배포
11. AWS ElasticCache 설정
11. AWS S3 설정
12. AWS CloudFront 설정
13. 소셜 로그인 설정

## 1. 프로젝트 구성도

## 기술 스택



## 시스템 아키텍처

## 2. Spring 서버 ec2 세팅

Server spec : t2.micro

os : Amazon Linux 2

docker & docker-compose 설치

```
# 서버 업데이트
sudo yum update

sudo yum upgrade

# 도커 설치
sudo yum install docker -y

# docker-compose 설치
sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

3. nginx 서버 ec2 세팅

Server spec : t2.micro

os : Amazon Linux 2

docker & docker-compose 설치

```
# docker 설치

sudo yum install docker -y

# nginx 설치

amazon-linux-extras list | grep nginx
sudo amazon-linux-extras install -y nginx1
nginx -v
sudo service nginx start
sudo systemctl status nginx
```

4. jenkins 서버 ec2 세팅

```
# Jenkins 설치

sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo systemctl start jenkins
sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key
sudo yum install jenkins

# jenkins 상태 확인 & 시작
sudo systemctl status jenkins
sudo systemctl start jenkins

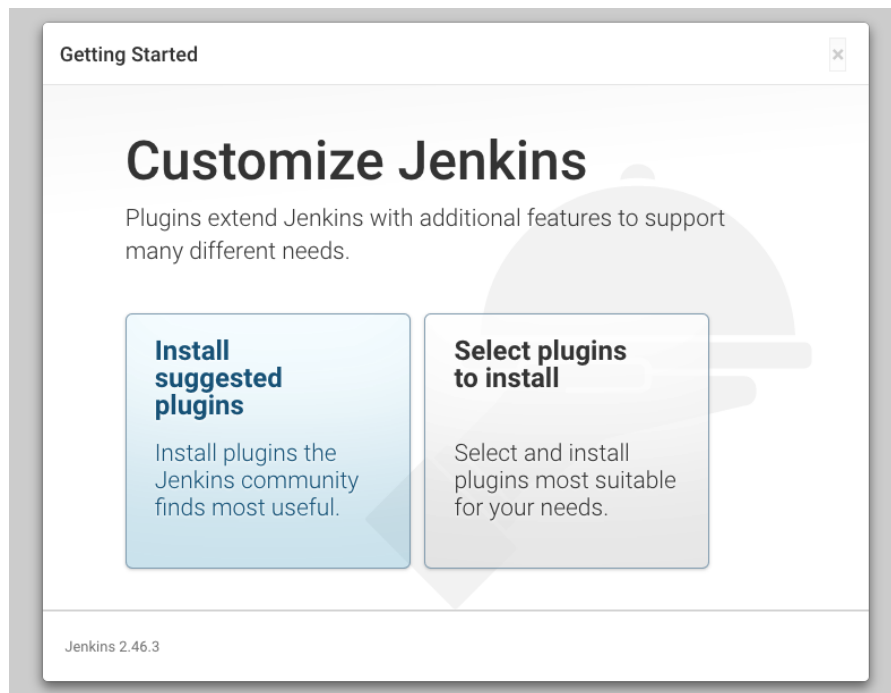
# jdk 설치
sudo yum install java-11-amazon-corretto

# 임시 비밀 번호 확인
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

임시 비밀번호 입력하고 설치를 계속한다.



```
# 임시 비밀 번호 확인
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```



추천 플러그인만 설치

### 필요 플러그인 설치

- [Generic Webhook Trigger Plugin](#)
- [GitLab](#)
- [Gitlab API Plugin](#)
- [GitLab Authentication plugin](#)
- [Mattermost Notification Plugin](#)
- [Publish Over SSH](#)

### Publish Over SSH 설정

```
# 키 발급
ssh-keygen -t rsa -b 4096 -m PEM
```

로 키를 만든다.

키를 만들때 추가 사항을 물어보는데 그냥 엔터 두번누른다

~/.ssh 경로에 있는 폴더에 들어가면 우리가 만든 키가 있다.

- id\_rsa : 개인키
- id\_rsa.pub : 공개키

```
# 키가 있는 디렉토리로 이동
cd ~/.ssh

# 공개키를 복사한다.
more id_rsa.pub
```

복사한 공개키를 nginx , spring server ~/.authorized\_keys에 복사해서 붙여 넣는다.

```
more id_rsa
```

먼저 Jenkins 설정 -> 시스템 설정에 들어가서 내리다 보면 Publish over SSH라는 항목이 보인다.

Publish over SSH

Jenkins SSH Key ?

Passphrase ?

Path to key ?

Key ?

☐ Disable exec ?

key에다가  
id\_rsa파일에 있는 키를 전부 복사해서 붙여넣자.

SSH Server

Name ?

! Required. Cannot contain < & ' " \

Hostname ?

! Required

Username ?

! Required

Remote Directory ?

고급...

Test Configuration

- name: 서버를 식별할 이름(Spring Server)
- hostname : 식별할 서버의 내부 IP
- Username : ec2-user
- Remote directory : /home/ec2-user

test configuration 눌러서 success가 나오면 연결 성공

마찬 가지로 openvidu 서버 nginx 서버 spring 서버 1 , spring 서버 2 총 4대의 대해서 서버 설정을 완료한다.

Mattermost notification 설정

## Global Mattermost Notifier Settings

Endpoint [?](#)

Channel [?](#)

Icon to use [?](#)

Build Server URL [?](#)

Mattermost Custom Proxy Settings...

Success

Test Connection

jenkins 시스템 구성 설정 맨밑에서 mattermost 설정

빌드시 mattermost 알람을 보내기 위해 endpoint에다가 mattermost 채널을 입력한다.

전체 Incoming Webhook 설정

[< Meeting! SSAFY\(으\)로 돌아가기](#)

😊 커스텀 이미지

🔗 통합

전체 Incoming Webhook

전체 Outgoing Webhook

슬래시 명령어

### 통합

[앱 디렉터리](#)를 방문하여 Mattermost에 대한 자체 호스팅인 third-party 앱 과 통합기능을 찾으십시오.



#### 전체 Incoming Webhook

Incoming Webhook은 외부 시스템에서 메시지를 받을 수 있게합니다.



#### 전체 Outgoing Webhook

Outgoing Webhook은 외부 시스템에 메시지를 보내고 응답받을 수 있게합니다.



#### 슬래시 명령어

슬래시 명령어는 외부에 연결한 서비스에 이벤트를 보냅니다.

제목

웹훅 설정 페이지에 대해 최대 64자의 제목을 지정합니다.

설명

웹훅에 대한 설명을 입력하세요.

채널

--- 채널을 선택하세요 ---

웹훅 페이로드를 수신할 기본 채널(공개 혹은 비공개)입니  
다. 비공개 채널로 웹훅을 설정할 때에는 그 채널에 속해있  
어야 합니다.

이 채널로 고정

☐

설정되면, 들어오는 웹훅은 선택된 채널에만 게시할 수 있  
습니다.

취소

저장

설정을 입력하고 만들어진 채널 url을 복사해서 jenkins 설정에 넣는다.

test connection Succes가 나오면 설정이 정상적으로 완료됐다.

파이프라인 생성

젠킨스에서 new 프로젝트 파이프라인 설정후 프로젝트를 생성한다

프로젝트 구성에서 파이프라인 스크립트를 작성한다.

```
pipeline {
  agent any
  environment {
    DOCKER_REPOSITORY = "suker800/color-the-rock"
  }

  stages {
    stage('Clone') {
      steps {
        git branch: 'backend_develop', credentialsId: 'GitLab-token', url: 'https://lab.ssafy.com/s08-webmobile1-sub2/S08P12A407'
      }
    }
    stage('Clean build'){
      steps{
        sh '''
        cd backend
        ls -al
        chmod +x ./gradlew
        sudo ./gradlew clean bootjar
        '''
      }
    }
    stage('Docker build'){
      steps{
        echo 'docker build'
        sh """ #!/bin/bash
        PREV_IMAGE='sudo docker images --filter=reference='suker800/' -q'
        echo "prev IMAGE : \${PREV_IMAGE}"
        if [[ -n \${PREV_IMAGE} ]]; then
          echo "prev image delete"
          sudo docker rmi \${PREV_IMAGE}
        fi
        sudo docker build . -t $DOCKER_REPOSITORY --no-cache
        sudo docker push $DOCKER_REPOSITORY
        """
      }
    }
    stage('Deploy Server 1 '){
      steps{
        sshPublisher(publishers: [sshPublisherDesc(configName: 'Server 1', transfers: [sshTransfer(cleanRemote: false, excludes: '', execCommand: ""
export DOCKER_REPOSITORY=$DOCKER_REPOSITORY
sh deploy.sh
""", execTimeout: 120000, flatten: false, makeEmptyDirs: false, noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: '', remoteDirectorySDF: false, removePrefix: '', sourceFiles: 'docker-comp
}
)
stage('Deploy Server 2 '){
      steps{
        sshPublisher(publishers: [sshPublisherDesc(configName: 'Server 2', transfers: [sshTransfer(cleanRemote: false, excludes: '', execCommand: ""
export DOCKER_REPOSITORY=$DOCKER_REPOSITORY
sh deploy.sh
""", execTimeout: 120000, flatten: false, makeEmptyDirs: false, noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: '', remoteDirectorySDF: false, removePrefix: '', sourceFiles: 'docker-comp
)
)
stage('Deploy Openvidu Server'){
      steps{
        sshPublisher(publishers: [sshPublisherDesc(configName: 'Openvidu Server', transfers: [sshTransfer(cleanRemote: false, excludes: '', execCommand: ''docker-compose down
docker pull suker800/color-the-rock:latest
docker rmi -f $(docker images -f "dangling=true" -q)
docker-compose up -d
''', execTimeout: 120000, flatten: false, makeEmptyDirs: false, noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: '', remoteDirectorySDF: false, removePrefix: '', sourceFiles: 'docker-comp
)
)
stage('Switch Nginx'){
```

```
steps{
    sshPublisher(publishers: [sshPublisherDesc(configName: 'Nginx Server', transfers: [sshTransfer(cleanRemote: false, excludes: '', execCommand: '', execTimeout: 120000, flatten: false, makeEmptyF
    )
}
stage('Mattermost notification') {
    steps {
        script {
            try {
                mattermostSend (
                    color: "#2A42EE",
                    message: "Build STARTED: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
                )
            } catch(e) {
                currentBuild.result = "FAILURE"
            } finally {
                if(currentBuild.result != "SUCCESS") {
                    mattermostSend (
                        color: "danger",
                        message: "Build FAILED: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
                    )
                } else {
                    mattermostSend (
                        color: "good",
                        message: "Build SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
                    )
                }
            }
        }
    }
}
```

파이프라인 순서 :

1. git 최신 내용을 내려받는다.
2. gradle을 이용해서 이전 빌드 파일들을 삭제하고 jar 파일을 빌드한다.
3. docker를 통해서 만들어진 jar파일을 기반으로 이미지를 빌드하고 docker-hub에 푸시한다.
4. 스프링 서버 1에서 이미지를 내려받고 서버를 다른 포트에 실행한다.
5. 스프링 서버 2에서 이미지를 내려받고 서버를 다른 포트에 실행한다.
6. openvidu 서버에
7. nginx 서버에서 스프링 서버가 잘 실행이 되어있는지 health 체크하고 정상적으로 실행이 되었다면 포트를 스위칭한다.
8. mattermost로 빌드가 정상적이라면 메시지 발송

jenkins gitlab 웹훅 설정

☒

Build when a change is pushed to GitLab. GitLab webhook URL: http://3.34.70.57:8080/project/Project%202 

?

Enabled GitLab triggers

☒ Push Events

☐ Push Events in case of branch delete

☒ Opened Merge Request Events

☐ Build only if new commits were pushed to Merge Request 

?

☐ Accepted Merge Request Events

☐ Closed Merge Request Events

Rebuild open Merge Requests

Never

▼

프로젝트 구성 중간에 Build when a change is pushed to gitlab. 메뉴 설정

webhook url을 따로 저장해둔다.

밑에 있는 Secret token generate해서 저장해둔다.

Secret token 

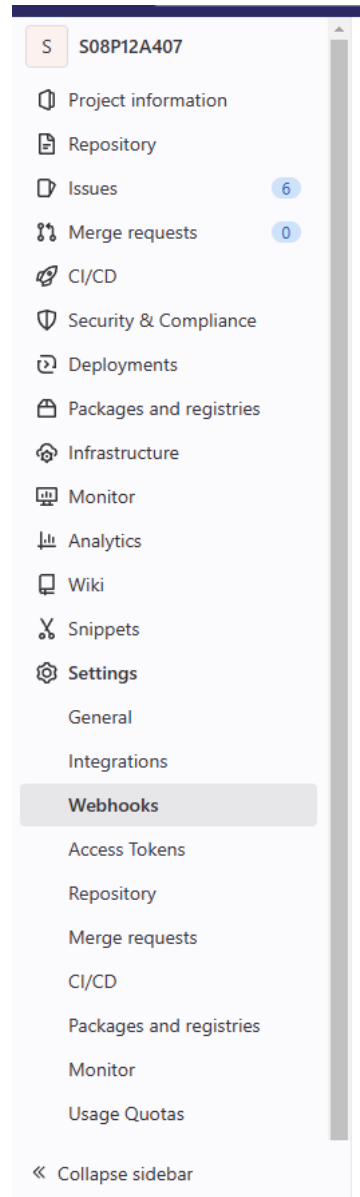
?

Generate

Clear

포팅 메뉴얼

7



빌드할 프로젝트에 들어가서 Settings - > Webhooks 메뉴로 들어간다.

## Webhooks

[Webhooks](#) enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

### URL

URL must be percent-encoded if it contains one or more special characters.

### Secret token

Used to validate received payloads. Sent with the request in the **X-Gitlab-Token** HTTP header.

### Trigger

☒ Push events

Push to the repository.

☐ Tag push events

A new tag is pushed to the repository.

☐ Comments

A comment is added to an issue or merge request.

☐ Confidential comments

A comment is added to a confidential issue.

☐ Issues events

An issue is created, updated, closed, or reopened.

☐ Confidential issues events

A confidential issue is created, updated, closed, or reopened.

☐ Merge request events

A merge request is created, updated, or merged.

☐ Job events

A job's status changes.

url에 복사해둔 gitlab webhook url을 넣고

secret token을 넣고 저장

테스트해서 200 메시지가 넘어오면 연결 성공

## 5. nginx ec2 설정

```
# nginx 설치
amazon-linux-extras list | grep nginx
```



```
sudo amazon-linux-extras install -y nginx1
nginx -v
```

nginx 설치 후 버전 확인

```
ssh-keygen -t rsa -b 4096 -m PEM
```

키 생성 후 공개키를 마찬가지로 Spring server 1,2 Openvidu 서버에 복사한다.

## 6. Spring 서버 배포 쉘스크립트 설정

```
#!/usr/bin/env bash

echo "> $DOCKER_REPOSITORY"
sudo true > RESULT
sudo chmod 666 /var/run/docker.sock
# 현재 사용하고 있는 포트와 유휴 상태인 포트를 체크한다.
RESPONSE=$(curl -s localhost:8080/actuator/health)
echo "> RESPONSE : "$RESPONSE

IS_ACTIVE=$(echo ${RESPONSE} | grep 'UP' | wc -l)
echo "> IS_ACTIVE "$IS_ACTIVE
if [ $IS_ACTIVE -eq 1 ];
then
    IDLE_PORT=8081
    IDLE_PROFILE=prod-green
    CURRENT_PORT=8080
    CURRENT_PROFILE=prod-blue
else
    IDLE_PORT=8080
    IDLE_PROFILE=prod-blue
    CURRENT_PORT=8081
    CURRENT_PROFILE=prod-green
fi

echo "> 다음 사용할 포트" $IDLE_PORT
echo "> 다음 사용할 프로필 " $IDLE_PROFILE

# 도커 허브에서 PULL을 한다.
docker pull $DOCKER_REPOSITORY
docker rm $(docker ps --filter status=exited -q)
docker rmi -f $(docker images -f "dangling=true" -q)

# 도커를 통해 컨테이너를 실행시킨다.

echo "> sudo nohup docker run -p $IDLE_PORT:8080 -e "USE_PROFILE=$IDLE_PROFILE" --env-file .env $DOCKER_REPOSITORY > nohup.out 2>&1 &"
sudo nohup docker run -p $IDLE_PORT:8080 --env-file .env -e "USE_PROFILE=prod" $DOCKER_REPOSITORY > nohup.out 2>&1 &

echo "> 60초동안 5초마다 Health Check"

for RETRY in {1..12}
do
    for i in {1..5} ;
    do
        echo "> Health Check까지 " $(( 6 - i))초 남음

        sleep 1
        done

        RESPONSE=$(curl -s localhost:${IDLE_PORT}/actuator/health)
        IS_ACTIVE=$(echo ${RESPONSE} | grep 'UP' | wc -l)

        if [ $IS_ACTIVE -ge 1 ]; then
            echo "> Health Check Success"
            echo "IDLE_PORT" $IDLE_PORT
            echo "$IDLE_PORT" > RESULT
            exit 0
        else
            echo "> Health Check Failed"
            echo "> Health Check RESPONSE : " $RESPONSE
        fi
    fi
if [ $RETRY -eq 10 ]; then
    echo "> Health Check Failed"
    echo "FAIL" > RESULT
fi
done

exit 1
```

셸 스크립트 설정

```
FROM openjdk:11
ARG JAR_FILE=backend/build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENV USE_PROFILE local
ENTRYPOINT ["java","-jar","-Dspring.profiles.active=${USE_PROFILE}","/app.jar"]
```

Dockerfile 설정

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: ${DATASOURCE_URL}
    username: ${DATASOURCE_USERNAME}
    password: ${DATASOURCE_PASSWORD}
  jpa:
    hibernate:
      ddl-auto: create
    show-sql: true
    database-platform: org.hibernate.dialect.MySQL8Dialect
  security:
    user:
      name: ${ADMIN_NAME}
      password: ${ADMIN_PASSWORD}
      roles: ADMIN
  oauth2:
```

```
client:
  registration:
    kakao:
      client-id: ${KAKAO_ID}
      client-secret: ${KAKAO_SECRET}
      redirect-uri: "https://colortherock.com/login/oauth2/code/kakao"
      authorization-grant-type: authorization_code
      client-authentication-method: POST
      client-name: kakao
      scope:
        - account_email
    google:
      client-id: ${GOOGLE_ID}
      client-secret: ${GOOGLE_SECRET}
      redirect-uri: "https://colortherock.com/login/oauth2/code/google"
      scope:
        - profile
        - 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

servlet:
  multipart:
    max-file-size: 100MB
    max-request-size: 100MB

redis:
  host: ${REDIS_HOST}
  port: ${REDIS_PORT}

cloud:
  aws:
    credentials:
      access-key: ${AWS_ACCESS_KEY}
      secret-key: ${AWS_SECRET_KEY}
    s3:
      bucket: ${BUCKET_NAME}
    region:
      static: ap-northeast-2
    stack:
      auto: false
  notification:
    mattermost:
      enabled: true # mmSender를 사용할 지 여부, false면 알림이 오지 않는다
      webhook-url: ${MM_WEBHOOK_URL} # 위의 Webhook URL을 기입
  report:
    mattermost:
      enabled: true
      webhook-url: ${MM_REPORT_URL}
      pretext: 신고 누락 발생 알림!
  springdoc:
    swagger-ui:
      disable-swagger-default-url: true
      path: /api/swagger-ui.html
    api-docs:
      path: /api/post-docs
  OPENVIDU_URL: ${OPENVIDU_URL}
  OPENVIDU_SECRET: ${OPENVIDU_SECRET}
  RECORDING_PATH: ${RECORDING_PATH}
  CLOUDFRONT_URL: ${CLOUDFRONT_URL}
```

배포 서버 application.yaml 파일

jenkins에서 설정한 환경변수로 spring server를 실행시킨다.

```
DOCKER_REPOSITORY=suker800/color-the-rock
DATASOURCE_URL= # db url
DATASOURCE_USERNAME= # db 유저 이름
DATASOURCE_PASSWORD= # 비밀번호
KAKAO_ID= # 카카오 client id
KAKAO_SECRET= # 카카오 oauth secret
GOOGLE_ID= # 구글 oauth client id
GOOGLE_SECRET= # 구글 oauth client secret
REDIS_HOST= # redis 경로
REDIS_PORT= # redis 포트
OPENVIDU_URL= # 오픈비두 서버 url
OPENVIDU_SECRET= # 오픈비두 서버 비밀번호
AWS_ACCESS_KEY= # aws 액세스 키
AWS_SECRET_KEY= # aws 시크릿 키
BUCKET_NAME= # 버킷 이름
RECORDING_PATH= # 녹화 경로
CLOUDFRONT_URL= # 클라우드front 경로
MM_WEBHOOK_URL= # mm 알림 url
ADMIN_NAME= # 관리자 계정 아이디
ADMIN_PASSWORD= # 관리자 계정 비밀번호
MM_REPORT_URL= # mm 신고 알림 url
```

aapplication.yaml 파일에 적용할 환경변수

## 7. nginx 배포 설정

```
#!/usr/bin/env bash
# 현재 사용중인 포트를 확인한다.
RESPONSE=$(curl -s -k -L colortherock.com/actuator/health)
echo "> RESPONSE : "$RESPONSE

IS_ACTIVE=$(echo ${RESPONSE} | grep 'UP' | wc -l)
echo "> IS_ACTIVE" "$IS_ACTIVE"
CURRENT_PORT=$(curl -k -L colortherock.com/port | grep 'BLUE' | wc -l)
echo " CURRENT PORT" "$CURRENT_PORT"
if [ "$IS_ACTIVE" -eq 1 ];
then
  if [ "$CURRENT_PORT" -eq 1 ];
  then
    IDLE_PORT=8081
    IDLE_PROFILE=GREEN
    CURRENT_PORT=8080
    CURRENT_PROFILE=BLUE
  else
    IDLE_PORT=8080
    IDLE_PROFILE=BLUE
    CURRENT_PORT=8081
    CURRENT_PROFILE=GREEN
  fi
```

```
else
    IDLE_PORT=8080
    IDLE_PROFILE=BLUE
    CURRENT_PORT=8081
    CURRENT_PROFILE=GREEN
fi

# 배포했던 서버들이 전부 성공적으로 빌드가 되었는지 확인한다.
SERVERS=(`cat servers`)
# 서버들의 개수
TOTAL_SERVER=${#SERVERS[@]}
# 임시로 Spring 서버들의 응답을 저장할 파일을 하나 만든다.
sudo true > servers_response
sudo chmod 664 servers_response

for server in ${SERVERS[*]} ; do
    # spring 서버들의 결과 파일들을 가져온다.
    # 결과 파일 가져온걸 servers_response에 저장
    sudo true > RESULT
    scp ${server}:RESULT .
    cat RESULT >> servers_response
done
echo "> SERVERS RESULT : $(cat servers_response)"

if [ "$TOTAL_SERVER" -eq "$(grep -c "$IDLE_PORT" servers_response)" ]; then
    echo "> 정상 배포 완료"
    echo "set \$active_server $IDLE_PROFILE;" | sudo tee /etc/nginx/default.d/port.conf
    echo "> nginx 재시작"
    sudo systemctl reload nginx
    # 스위칭 하기위해 spring 서버들의 docker 컨테이너를 내리자.
    for server in ${SERVERS[*]} ; do
        echo "> server : "$server
        echo "> ssh" $server docker kill "$(ssh $server docker ps -qf publish=$CURRENT_PORT)"
        ssh $server docker kill "$(ssh $server docker ps -qf publish=$CURRENT_PORT)" 2> /dev/null || echo "현재 실행중인 서버가 없습니다. CURRENT_PORT: $CURRENT_PORT"
    done
else
    echo "> 배포 실패"
    echo "스위칭 하려고 켜놓은 서버 포트들을 전부 종료합니다."
    for server in ${SERVERS[*]} ; do
        echo "> server : "$server
        echo "> ssh" $server docker kill "$(ssh $server docker ps -qf publish=$IDLE_PORT)"
        ssh $server docker kill "$(ssh $server docker ps -qf publish=$IDLE_PORT)" 2> /dev/null || echo "현재 실행중인 서버가 없습니다. CURRENT_PORT: $IDLE_PORT"
    done
done
fi
```

nginx에서 배포 스크립트 설정

```
upstream BLUE{

    server 스프링 서버 1 ip 주소:8080 weight=100 max_fails=3 fail_timeout=3s;
    server 스프링 서버 2 ip 주소:8080 weight=100 max_fails=3 fail_timeout=3s;
}
upstream GREEN{

    server 스프링 서버 1 ip 주소:8081 weight=100 max_fails=3 fail_timeout=3s;
    server 스프링 서버 2 ip 주소:8081 weight=100 max_fails=3 fail_timeout=3s;
}
```

nginx.conf에 upstream을 각각 설정한다.

```
set $active_server BLUE; # /default.d/port.conf
```

/default.d/port.conf

현재 포트 변수를 설정하는 파일

```
location / {
    root /usr/share/nginx/html;
    index index.html;
    try_files $uri $uri/ /index.html;
}
```

/default.d/port.conf

리액트 정적파일을 가져오는 location 설정

```
upstream ALL_BLUE{
    server 스프링 서버 1 ip 주소:8080 weight=100 max_fails=3 fail_timeout=3s;
    server 스프링 서버 2 ip 주소:8080 weight=100 max_fails=3 fail_timeout=3s;
    server 오픈비두 서버 ip 주소:8080;
}
```

/conf.d/all\_server\_blue.conf

모든 서버 blue 포트 upstream

```
upstream ALL_GREEN{
    server 스프링 서버 1 ip 주소:8081 weight=100 max_fails=3 fail_timeout=3s;
    server 스프링 서버 2 ip 주소:8081 weight=100 max_fails=3 fail_timeout=3s;
    server 오픈비두 서버 ip 주소:8080;
}
```

/conf.d/all\_server\_green.conf

모든 서버 green 포트 upstream

```
upstream openvidu{
    server 오픈비두 서버 ip 주소:8080;
}
```

```

server {
    server_name  colorthero.ck.com;
    root         /usr/share/nginx/html;

    # Load configuration files for the default server block.

    include /etc/nginx/default.d/*.conf;

    location ~* ^/(oauth2|login){
        proxy_pass http://$active_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
    location = /actuator/health {
        proxy_pass http://$active_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    location = /api/refresh {
        proxy_pass http://$active_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    location ~ ^/(api/live/(.+)/recording/start|api/live/uploadRecord|api/live/(.+)/recording/list)$ {

        proxy_pass http://openvidu;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;

    }

    location ~* ^/(api) {
        proxy_pass http://ALL_$active_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
    location /nginx_status{
        stub_status on;
        access_log off;
        allow 127.0.0.1;
        deny all;
    }
    location = /port {
        add_header Content-Type text/plain;
        if ($remote_addr = "nginx ip 주소"){
            return 200 '$active_server';
        }
        deny all;
    }
}

```

nginx.conf 설정파일 server 블록

nginx https 적용 순서대로 따라친다.

```

sudo snap install certbot --classic
sudo wget -r --no-parent -A 'epel-release-*.rpm' http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/e/
sudo rpm -Uvh dl.fedoraproject.org/pub/epel/7/x86_64/Packages/e/epel-release-*.rpm
sudo yum-config-manager --enable epel*
sudo yum install -y certbot python2-certbot-apache
sudo yum install certbot-nginx
sudo certbot --nginx

```

## 8. 프론트엔드 리액트 프로젝트 배포

새로운 프로젝트 freestyle로 설정해서 새로 생성

Git ?

Repositories ?

Repository URL ?

https://lab.ssafy.com/s08-webmobile1-sub2/S08P12A407

Credentials ?

suker800@gmail.com/\*\*\*\*\*

+ Add

고급...

Add Repository

git 프로젝트 설정

☒ Build periodically ?

Schedule ?

H 8 \* \* \*

Would last have run at Tuesday, February 14, 2023 at 8:27:17 AM Korean Standard Time; would next run at Wednesday, February 15, 2023 at 8:27:17 AM Korean Standard Time.

매일 아침 8시 27분에 자동 빌드

```
# Use the official Node.js image as the base image
FROM node:18.13-alpine

# Set the working directory
WORKDIR /app

# Copy the package.json and package-lock.json files
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application files
COPY . .

ENV GENERATE_SOURCEMAP=false
ENV NODE_OPTIONS="--max-old-space-size=4096"
# Build the React project
RUN npm run build

# Specify the command to run when the container starts
CMD [ "npm", "run", "serve" ]
```

리액트 Dockerfile 설정

```
cd frontend/color-the-rock-app # 프론트 디렉토리로 이동
rm -rf build* # 기존 빌드파일 삭제
sudo chmod 666 /var/run/docker.sock
QUIT_CONTAINER=$(docker ps --filter status=exited -q)
if [ -n "$QUIT_CONTAINER" ]
then
    docker rm $QUIT_CONTAINER
fi
DANGLING_IMAGE=$(docker images -f "dangling=true" -q)
if [ -n "$DANGLING_IMAGE" ]
then
    docker rmi -f $DANGLING_IMAGE # none 태그 이미지 삭제
fi

docker build . -t react-alpine # react 이미지 빌드
CONTAINER=$(docker run -d -p 3000:3000 react-alpine npm run start) # 컨테이너 실행
docker cp $CONTAINER:/app/build ./build # 컨테이너 안에 있는 빌드 파일로 로컬로 복사
docker kill $(docker ps -f ancestor=react-alpine -q) # 도커 컨테이너 종료
tar cvf build.tar ./build # 배포하기 쉽게 tar로 묶는다.
```

프론트 배포 스크립트

SSH Server

Name ?

Nginx Server

고급...

Transfers

Transfer Set

Source files ?

frontend/color-the-rock-app/build.tar

Remove prefix ?

frontend/color-the-rock-app

Remote directory ?

Exec command ?

```
sudo tar xvf build.tar
sudo cp -f -r ./build/* /usr/share/nginx/html/
sudo systemctl reload nginx
```

All of the transfer fields (except for Exec timeout) support substitution of [Jenkins environment variables](#)

고급...

빌드 파일을 nginx에 전송해야 한다.

buid.tar 파일을 nginx 서버에 이동시키고

압축을 푼다음 nginx를 재시작 한다.

## 9. Openvidu 서버 배포 설정

```
# docker 설치
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io

# docker-compose 설치
sudo apt-get install docker-compose-plugin
```

openvidu 서버 도커 설치

```
# openvidu 설치
cd /opt
curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh | bash
sudo curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh | bash
```

openvidu를 실행하기 위해 docker-compose로 서버 실행

```
version: "3"

services:
  spring:
    build: .
    restart: always
    ports:
      - 8080:8080
    env_file:
      - .env
    image: suker800/color-the-rock:latest
    environment:
      - USE_PROFILE=prod
      - REDIS_HOST=host
    volumes:
      - /opt/openvidu/recordings:/opt/openvidu/recordings
  redis:
    image: redis:alpine
    ports:
      - 6379:6379
```

```
# 실행중인 서버 down
docker-compose down
# 새 이미지 pull
docker pull suker800/color-the-rock:latest
```

```
# 이전 이미지 삭제
docker rmi -f $(docker images -f "dangling=true" -q)
# 백그라운드로 서버 다시 실행
docker-compose up -d
```


## 10. RDS 배포

RDS 생성 페이지에서 MySQL 선택


엔진 옵션

엔진 유형 정보


☐ Amazon Aurora




☒ MySQL




☐ MariaDB




☐ PostgreSQL



☐ Oracle



☐ Microsoft SQL Server



DB 인스턴스 식별자 정보

DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.

database-1

DB 인스턴스 식별자는 대소문자를 구분하지 않지만 'mydbinstance'와 같이 모두 소문자로 저장됩니다. 제약: 1~60자의 영숫자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자여야 합니다. 하이픈 2개가 연속될 수 없습니다. 하이픈으로 끝날 수 없습니다.

▼ 자격 증명 설정

마스터 사용자 이름 정보

DB 인스턴스의 마스터 사용자에 로그인 ID를 입력하세요.

admin

1~16자의 영숫자. 첫 번째 문자는 글자여야 합니다.

☐ AWS Secrets Manager에서 마스터 보안 인증 관리 - 신규

Secrets Manager에서 마스터 사용자 보안 인증을 관리합니다. RDS는 사용자 대신 암호를 생성하고 수명 주기 동안 이를 관리할 수 있습니다.

☐ 암호 자동 생성

Amazon RDS에서 사용자를 대신하여 암호를 생성하거나 사용자가 직접 암호를 지정할 수 있습니다.

마스터 암호 정보

제약 조건: 8자 이상의 인쇄 가능한 ASCII 문자. 다음은 포함할 수 없습니다. /(슬래시), '(작은따옴표)', "(큰따옴표) 및 @(앳 기호).

마스터 암호 확인 정보

RDS 비밀번호와 유저 이름 설정

컴퓨팅 리소스

이 데이터베이스의 컴퓨팅 리소스에 대한 연결을 설정할지를 선택합니다. 연결을 설정하면 컴퓨팅 리소스가 이 데이터베이스에 연결할 수 있도록 연결 설정이 자동으로 변경됩니다.

☒ EC2 컴퓨팅 리소스에 연결 안 함

이 데이터베이스의 컴퓨팅 리소스에 대한 연결을 설정하지 않습니다. 나중에 컴퓨팅 리소스에 대한 연결을 수동으로 설정할 수 있습니다.

☐ EC2 컴퓨팅 리소스에 연결

이 데이터베이스의 EC2 컴퓨팅 리소스에 대한 연결을 설정합니다.

Virtual Private Cloud(VPC) 정보

VPC를 선택합니다. VPC는 이 DB 인스턴스의 가상 네트워킹 환경을 정의합니다.

Default VPC (vpc-04faef7edc0d6bbe1)

해당 DB 서버넷 그룹이 있는 VPC만 나열됩니다.

데이터베이스를 생성한 후에는 VPC를 변경할 수 없습니다.

DB 서버넷 그룹 정보

DB 서버넷 그룹을 선택합니다. DB 서버넷 그룹은 선택한 VPC에서 DB 인스턴스가 어떤 서버넷과 IP 범위를 사용할 수 있는지를 정의합니다.

default-vpc-04faef7edc0d6bbe1

퍼블릭 액세스 정보

☐ 예

RDS는 데이터베이스에 퍼블릭 IP 주소를 할당합니다. VPC 외부의 Amazon EC2 인스턴스 및 다른 리소스가 데이터베이스에 연결할 수 있습니다. VPC 내부의 리소스도 데이터베이스에 연결할 수 있습니다. 데이터베이스에 연결할 수 있는 리소스를 지정하는 VPC 보안 그룹을 하나 이상 선택합니다.

☒ 아니요

RDS는 퍼블릭 IP 주소를 데이터베이스에 할당하지 않습니다. VPC 내부의 Amazon EC2 인스턴스 및 다른 리소스만 데이터베이스에 연결할 수 있습니다. 데이터베이스에 연결할 수 있는 리소스를 지정하는 VPC 보안 그룹을 하나 이상 선택합니다.

VPC 보안 그룹(방화벽) 정보

데이터베이스에 대한 액세스를 허용할 VPC 보안 그룹을 하나 이상 선택합니다. 보안 그룹 규칙이 적절한 수신 트래픽을 허용하는지 확인합니다.

☒ 기존 항목 선택

기존 VPC 보안 그룹 선택

☐ 새로 생성

새 VPC 보안 그룹 생성

## VPC와 보안그룹, 퍼블릭 액세스 설정

VPC는 프로젝트용이라 default로 사용하였다.

퍼블릭 액세스는 아니요 설정

RDS 주소를 jenkins 서버 DATASOURCE\_URL 변수에 넣는다.

## 11. AWS ElasticCache 설정

### 클러스터 생성 방법 선택

다음 옵션 중 하나를 선택하여 새 클러스터를 생성합니다.

☒ 새 클러스터 구성 및 생성  
새 클러스터의 모든 구성 옵션 설정

☐ 백업에서 복원  
기존 백업 또는 .rdb 파일을 사용하여 클러스터를 복원합니다.

### 클러스터 모드

가동 중단 없이 클러스터 크기를 동적으로 조정합니다.

☐ 활성화됨  
클러스터 모드에서는 확장성과 가용성을 개선하도록 여러 샤드에서 복제할 수 있습니다.

☒ 비활성화됨  
Redis 클러스터에는 프라이머리 노드 하나와 최대 5개의 읽기 전용 복제본을 포함하는 단일 샤드(노드 그룹)가 있습니다.

☒ 클러스터 모드를 사용 중지하는 경우 샤드 수를 변경할 수 없습니다. 구성은 모든 Redis 명령과 기능을 지원하지  
만 최대 캐시 크기와 성능을 제한합니다. [자세히 알아보기](#)

단순 키 저장용이라 클러스터 모드는 비활성화 했다.

클러스터 모드를 하려면 Spring redis 설정을 바꾸어야 한다.





완료가 됐다면 버킷 생성

## 12 . AWS CloudFront 설정

cloudfront 생성 페이지로 이동한다.

원본

원본 도메인

AWS 원본을 선택하거나 사용자 원본의 도메인 이름을 입력합니다.

Q

color-the-rock.s3.ap-northeast-2.amazonaws.com

X

원본 경로 - 선택 사항 | 정보

원본 요청의 원본 도메인 이름에 추가할 URL 경로를 입력합니다.

원본 경로 입력

이름

이 원본의 이름을 입력합니다.

color-the-rock.s3.ap-northeast-2.amazonaws.com

S3 버킷 액세스 | 정보

☒ 공개

버킷은 공개 액세스를 허용해야 합니다.

☐ 원본 액세스 제어 설정(권장)

버킷은 CloudFront에 대한 액세스만 제한할 수 있습니다.

☐ Legacy access identities

CloudFront 원본 액세스 ID(OAI)를 사용하여 S3 버킷에 액세스합니다.

사용자 정의 헤더 추가 - 선택 사항

CloudFront는 원본으로 보내는 모든 요청에 이 헤더를 포함합니다.

헤더 추가

기준이 될 S3 스토리지를 선택한다.

## 13. 소셜 로그인 설정

카카오 (Kakao)

1. <https://developers.kakao.com/> 로 접속한다
2. 내 애플리케이션 > 애플리케이션 추가하기 > 앱 이름 입력 > 사업자명 입력 > 저장
3. 좌측 Nav 바에서 앱 설정 > 요약 정보 > 앱 키 > REST API키 제품 설정 > 카카  
오 로그인 > 보안 > Client Secret의 코드 발급 기록해둡니다.