AniTime

Ssafy 공통 프로젝트

07.3 – 8.18

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **기술 스택** | **기술 상세** | **버전** |
| 공통 | 형상 관리 | GitLab | - |
| 진척 관리 | Jira | - |
| 협업 툴 | Notion, Mattermost | - |
| BackEnd | Java | Openjdk 11 | 11.0.15.9 |
| Build | Gradle | 8.1.1 |
| Spring | SpringBoot | 2.7.13 |
| Spring Security | - |
| JWT | 0.11.5 |
| Spring-Data-JPA | - |
| WebSocket | - |
| OpenVidu | OpenVidu-Java-Client | 2.25.0 |
| Cloud Storage | AWS S3 | - |
| API Docs | Swagger2 | 3.0.0 |
| DB | DBMS | MariaDB | 10.3.23 |
| FrontEnd | Build | Node.js | 18.16.1 |
| React | React | 18.2.0 |
| Redux | 4.2.1 |
| Mui-material | 5.14.2 |
| Styled-Components | 6.0.5 |
| SockJS | 1.6.1 |
| Webstomp-client | 1.2.6 |
| OpenVidu | OpenVidu-browser | 2.28.0 |
| Infra | 서버 | AWS EC2 | - |
| Ubuntu | 20.04.6 |
| Nginx | 1.18.0 |
|  | (해.줘.) |  |  |
|  | 도커 |  |  |
|  | Ci/cd |  |  |
|  | Openvidu |  |  |
|  | rabbitmq |  |  |
|  | 등등 |  |  |
|  |  |  |  |

1. 인프라구성

* 1. Ubuntu 초기 설정

주의 사항 : 설치하며 작성되는 포트들은 모두 sudo ufw allow [portnumber]로 방화벽 해제를 진행

* 1. Docker 설치

sudo apt-get update

sudo apt-get install -y \

ca-certificates \

curl \

gnupg \

lsb-release

# GPG Key 인증

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

# docker repository 등록

$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $ (lsb\_release -cs) stable"

# 도커 설치

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io

# 도커 확인

sudo service docker status

* 1. Openvidu 설치

sudo apt-get update

sudo apt-get upgrade

sudo su

Openvidu 설치를 위한 가이드 경로로 이동

cd /opt

Openvidu 설치

curl <https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install\_openvidu\_latest.sh> | bash

cd openvidu

초기 ssh설치를 위한 기본 세팅

$ vi .env

.env 내용

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# OpenVidu configuration

# ----------------------

# 도메인 또는 퍼블릭IP 주소

DOMAIN\_OR\_PUBLIC\_IP=[도메인주소]

# 오픈비두 서버와 통신을 위한 시크릿

OPENVIDU\_SECRET=HOMEDONG

# Certificate type

CERTIFICATE\_TYPE=letsencrypt

# 인증서 타입이 letsencrypt일 경우 이메일 설정

LETSENCRYPT\_EMAIL=user@example.com

# HTTP port

HTTP\_PORT=80

# HTTPS port(해당 포트를 통해 오픈비두 서버와 연결)

HTTPS\_PORT=443

Esc -> :wq enter

./openvidu start

* + 1. Openvidu 포트 변경

\*가장 먼저 Openvidu부터 세팅이 되어야한다 이유는 Openvidu on promise install 가이드에서 nginx를 설치 후 설치하게 되면 버그가 있단 것을 알 수 있다.

$ docker ps

//docker ps를 통해 openvidu 설치확인

$./openvidu stop

$vi .env

.env에서 기존 사용 예정인 openvidu 전용 포트 번호로 수정

.env

# HTTP port

HTTP\_PORT=[personal port]

# HTTPS port(해당 포트를 통해 오픈비두 서버와 연결)

HTTPS\_PORT=[personal port]

./openvidu start

* 1. Jenkins 설치

참고 : https://velog.io/@chang626/Jenkins

sudo apt-get update

sudo apt-get install apache2

sudo apt-get install openjdk-11-jdk

curl -fsSL https://pkg.jenkins.io/debian/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 binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt update

sudo apt install Jenkins

//Jenkins 포트 번호 변경을 위한 경로

sudo vi /usr/lib/systemd/system/jenkins.service

// Jenkins\_port=[희망 Port번호]를 작성 후 저장

Sudo systemctl restart Jenkins

* 1. Jenkins CI/CD 구축

- FrontEnd

Item 중 pipeLine으로 설정하여 Item 생성

Config 중 Pipeline 내부에 스크립트 작성  
FrontEnd pipeline Script

pipeline {

agent any // 사용 가능한 에이전트에서 이 파이프라인 또는 해당 단계를 실행

stages {

stage('Prepare') {

steps {

git branch: ‘해당 브런치’,

url: 'token기반 gitlab URL’

}

post {

success {

sh 'echo "Successfully Cloned Repository"'

}

failure {

sh 'echo "Fail Cloned Repository"'

}

}

}

stage('Npm install'){

steps{

dir('./frontend'){

sh 'sudo npm install'

}

}

post {

success {

sh 'echo "Successfully Install Npm"'

}

failure {

sh 'echo "Fail Install Npm"'

}

}

}

stage('Build') {

steps {

// 여기서는 이제부터 npm test를 진행해주고

dir("./frontend"){

sh 'sudo npm run build'

}

}

post {

success {

echo 'npm build success'

}

failure {

echo 'npm build failed'

}

}

}

stage('Docker Rm') {

steps{

script{

try {

sh 'echo "Docker Rm Start"'

sh """

sudo docker stop docker-jenkins-pipeline-react

sudo docker rm docker-jenkins-pipeline-react

sudo docker rmi -f front/docker-jenkins-pipeline-react

"""

} catch (e) {

sh 'echo Docker image not found'

}

}

}

}

stage('Dockerizing'){

steps{

sh 'echo " Image Bulid Start"'

dir('./frontend'){

sh 'sudo docker build -t front/docker-jenkins-pipeline-react .'

}

}

post {

success {

sh 'echo "Bulid Docker Image Success"'

}

failure {

sh 'echo "Bulid Docker Image Fail"'

}

}

}

stage('Deploy') {

steps {

sh 'sudo docker run --name docker-jenkins-pipeline-react -d -e TZ=Asia/Seoul -p 3000:3000 front/docker-jenkins-pipeline-react '

}

post {

success {

echo 'success'

}

failure {

echo 'failed'

}

}

}

}

}

- BackEnd

Item 중 pipeLine으로 설정하여 Item 생성

Config 중 Pipeline 내부에 스크립트 작성  
BackEnd pipeline Script

pipeline {

agent any // 사용 가능한 에이전트에서 이 파이프라인 또는 해당 단계를 실행

stages {

stage("clone") {

steps {

git branch: “gitbranchname",

url: "giturl"

}

post {

success {sh "echo Successfully Cloned Repository";}

failure {sh "echo 'Fail Cloned Repository'";}

}

}

stage("Build") {

steps {

// gralew이 있어야됨. git clone해서 project를 가져옴.

dir("./backend"){

sh "chmod +x ./gradlew"

sh "./gradlew clean build"

sh "ls -al ./build"

}

}

post {

success {echo "gradle build success"}

failure {echo "gradle build failed"}

}

}

stage("Docker Rm") {

steps{

script{

try {

sh "echo 'Docker Rm Start'"

sh """

sudo docker stop docker-jenkins-pipeline-spring

sudo docker rm docker-jenkins-pipeline-spring

sudo docker rmi -f back/docker-jenkins-pipeline-spring

"""

} catch (e) {sh "echo Docker image not found"}

}

}

}

stage("Dockerizing"){

steps{

sh "echo Image Bulid Start"

dir("./backend"){

sh "sudo docker build -t back/docker-jenkins-pipeline-spring ."

}

}

post {

success {

sh "echo Bulid Docker Image Success"

}

failure {

sh "echo Bulid Docker Image Fail"

}

}

}

stage("Deploy") {

steps {

sh "sudo docker run --name docker-jenkins-pipeline-spring -d -e TZ=Asia/Seoul -p 8000:8000 back/docker-jenkins-pipeline-spring " }

post {

success { echo "success" }

failure { echo "failed" } } } }}

* 1. Nginx 구축 및 SSL 적용

1. # 설치
2. sudo apt-get install nginx
3. # 설치 확인 및 버전 확인
4. nginx -v
5. sudo apt-get install letsencrypt
6. sudo systemctl stop nginx
7. sudo letsencrypt certonly --standalone -d www제외한 도메인 이름
8. Sudo systemctl restart nginx
10. Cd /etc/nginx/sites-enabled
12. V vi (free).conf

(free).conf 작성

1. server {
2. listen 80 default\_server;
3. listen [::]:80 default\_server;
4. server\_name 도메인 주소;
5. location /api {
6. proxy\_pass (각자의 도메인:각자 포트);
7. proxy\_redirect default;
8. }
9. location /{
10. return 301 https://$server\_name$request\_uri;
11. }
12. location /ws{
13. proxy\_pass (각자의 도메인:각자 포트);
14. proxy\_set\_header Upgrade $http\_upgrade;
15. proxy\_set\_header Connection "upgrade";
16. proxy\_http\_version 1.1;
17. proxy\_set\_header Host $host;
18. proxy\_cache\_bypass $http\_upgrade;
19. }
20. }
21. server {
22. listen 8081 ssl;
23. listen [::]:8081 ssl;
24. server\_name (각자의 도메인:각자 포트);
25. ssl\_certificate /etc/letsencrypt/live/도메인주소/fullchain.pem; # managed by Certbot
26. ssl\_certificate\_key /etc/letsencrypt/live/도메인주소/privkey.pem; # managed by Certbot
27. location /ws{
28. proxy\_read\_timeout 300s;
29. proxy\_connect\_timeout 75s;
30. proxy\_pass (각자의 도메인:각자 포트);
31. proxy\_http\_version 1.1;
32. proxy\_set\_header Upgrade $http\_upgrade;
33. proxy\_set\_header Connection "upgrade";
34. proxy\_redirect off;
35. proxy\_set\_header Host $host;
36. proxy\_set\_header X-Real-IP $remote\_addr;
37. proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;
38. proxy\_set\_header X-Forwarded-Host $server\_name;
39. }
40. }
41. server {
42. listen 443 ssl;
43. listen [::]:443 ssl;
44. server\_name 도메인주소 도메인주소;
45. ssl\_certificate /etc/letsencrypt/live/도메인 주소/fullchain.pem; # managed by Certbot
46. ssl\_certificate\_key /etc/letsencrypt/live/도메인주소/privkey.pem; # managed by Certbot
47. location /api {
48. proxy\_pass (각자의 도메인:각자 포트);;
49. proxy\_redirect default;
50. proxy\_set\_header Host $http\_host;
51. proxy\_set\_header X-Real-IP $remote\_addr;
52. proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;
53. proxy\_set\_header X-Forwarded-Proto $scheme;
54. }
55. location /ws{
56. proxy\_read\_timeout 300s;
57. proxy\_connect\_timeout 75s;
58. proxy\_pass (각자의 도메인:각자 포트);;
59. proxy\_http\_version 1.1;
60. proxy\_set\_header Upgrade $http\_upgrade;
61. proxy\_set\_header Connection "upgrade";
62. proxy\_set\_header Origin "";
63. proxy\_set\_header X-Real-IP $remote\_addr;
64. proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;
65. proxy\_set\_header X-Forwarded-Host $server\_name;
66. }
67. location /{
68. proxy\_pass (각자의 도메인:각자 포트);
69. proxy\_redirect default;
70. proxy\_http\_version 1.1;
71. proxy\_set\_header Upgrade $http\_upgrade;
72. proxy\_set\_header Connection "upgrade";
73. proxy\_set\_header Host $http\_host;
74. proxy\_set\_header X-Real-IP $remote\_addr;
75. proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;
76. proxy\_set\_header X-Forwarded-Proto $scheme;
77. }
78. # include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
79. # ssl\_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
80. }
81. Sudo systemctl restart nginx
82. 1. Rabbitmq 구축

$sudo docker run -d --name rabbitmq -p 3333:5672 -p 44444:15672 -p 61613:61613 --restart=unless-stopped -e RABBITMQ\_DEFAULT\_USER=moi -e RABBITMQ\_DEFAULT\_PASS=moi rabbitmq:management

$sudo docker exec {이미지 ID} rabbitmq-plugins enable rabbitmq\_stomp

$sudo docker restart ‘containerID’