E201 포팅 메뉴얼

목차

1. **개발 환경**
   - **1-1. Frontend**
   - **1-2. Backend**
   - **1-3. Server**
   - **1-4. DB**
   - **1-5. 형상/이슈 관리**
2. **EC2 Settings**
   - **2-1. SSL 인증서 발급**
   - **2-2. EC2 포트 정리**
   - **2-3. Docker 및 Docker-compose 설치**
   - **2-4. Openvidu 설치 및 실행**
   - **2-5. Nginx 설치 및 설정**
   - **2-6. Whisper 설치**
   - **2-7. BERT Model 다운로드**
   - **2-8. Jenkins 루트 폴더 생성 및 Jenkins.war 다운로드**
   - **2-9. AWS S3 계정 연결**
3. **Deploy**
   - **3-1. Frontend (Dockerfile)**
   - **3-2. Backend (Dockerfile)**
   - **3-3. Docker-Compose (Frontend, Backend)**
   - **3-4. CI/CD (Jenkins, MySQL, Flask)**
   - **3-5. Shell Script**
4. **Jenkins Settings**
   - **4-1. Setting Acounts**
   - **4-2. TroubleShootings for install plugins**
   - **4-3. Check SecurityRealm**
   - **4-4. 플러그인 설치**
   - **4-5. DockerHub, Gitlab 연동**

1. 개발 환경

1.1. Frontend

- Node.js 22.5.1(LTS)
- React 18.3.1
  - Redux 9.1.2
  - Reduxjs/Toolkit 2.2.6
  - Router 6.25.1
- axios 1.7.2
- animejs 3.2.2
- socket-io: 4.7.5
- Openvidu Browser 2.30.1
- jwt-decode 4.0.0
- styled-components 6.1.12
- websocket 1.0.35

1.2. Backend

- Spring Boot
  - Spring Web
  - Spring Security
  - OAuth2 Client
  - Spring cloud aws starter S3
- Openvidu-java-client 2.30.0
- DB
  - Spring Data JPA
  - MySQL Driver
  - AWS Cloud S3
- Python: 3.8-slim
  - Torch: 1.6.0
  - Keras: 2.8.0
  - Numpy: 1.24.4
  - werkzeug: 2.0.2

    - huggingface-hub: 0.24.5
- OpenAI
    - ChatGPT: gpt-4o-mini
    - Whisper: LTS
- Utill
    - Lombok
    - Validation
    - Json

1.3. Server

- Ubuntu 20.04 LTS
- Nginx 1.18.0
- Docker 27.0.3
- Docker Compose 2.28.1
- OpenVidu 2.30.0
- Jenkins 2.45.2.3
- Flask 3.8-slim

1.4. DB

- MySQL 8.0
- Amazon S3

1.5. 형상 / 이슈 관리

- Jira
- GitLab

2. EC2 Settings

2.1. SSL 인증서 발급

```
# 1. certbot 설치
sudo apt install certbot
```

```
# 2. 웹서버 일시 중단 (없으면 생략)
# 2-1. nginx
sudo systemctl stop nginx

# 2-2. docker container
docker ps -s
docker stop [container id || container name]

# 3. 인증서 발급
sudo certbot certonly --standalone -d {server-domain.com}
www버전까지 포함하는 경우
sudo certbot certonly --standalone -d {server-domain.com} -d {www.serv
er-domain.com}
* ip 주소 확인
nslookup {server-domain.com}
```

2.2. EC2 포트 정리

포트번호	내용
22	SSH
80	HTTP (HTTPS 로 redirect)
443	HTTPS
3000	frontend (Docker)
8081	backend (Docker)
8082	backend (Flask)
3478	OpenVidu (TURN/STUN)
8443	OpenVidu (Media Server)
3306	DB - MySQL (Docker)
8081	Jenkins
40000:57000	Kurento Media Server
57001:65535	Kurento Media Server

2.2.1. ufw 방화벽 설정

```
# 1. ufw 연결 상태 확인
sudo ufw status

# 2. ufw 활성화 & 비활성화
sudo ufw enable
sudo ufw disable

# 3. 사용할 포트 허용하기
```

```
sudo ufw allow {port}
sudo ufw allow {port1 - port2}/<tcp/udp>

# 4. 허용된 포트 목록 확인
sudo ufw show added

# 5. ufw 상태 및 등록된 rule 확인하기
sudo ufw status numbered

# 6. 허용된 포트 삭제 후 적용
sudo ufw delete {port}
sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n
```

2.3. Docker 및 Docker-compose 설치

```
sudo yum update -y
sudo amazon-linux-extras install docker
sudo service docker start
sudo usermod -a -G docker ec2-user
sudo curl -L "https://github.com/docker/compose/releases/download/1.2
9.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-c
ompose
sudo chmod +x /usr/local/bin/docker-compose
```

2.4. OpenVidu 설치 및 실행

ref) https://docs.openvidu.io/en/stable/deployment/ce/on-premises/

개발 당시 OpenVidu v3이 Beta 버전이 출시되어 있었으나, 안정성을 위해 2.30.0 버전으로 개발 및 배포를 진행함.

⚠ Vidu 포트 변경전 NGINX 가 실행 중일 경우, 종료 후 작업 진행

→ 포트 충돌로 인해 정상적으로 인증서 발급이 안될 수 있음.

⚠ openvidu와 관련된 docker image / container가 없어야 함.

→ 버전이 맞지 않을 경우, 에러가 발생할 수 있음.

```
# 1. OpenVidu 서버 생성
sudo su
cd /opt
curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvi
```

```
du_latest.sh | bash
cd openvidu

# 2. Vidu Container SSL 인증서 발급
# 2-1. .env파일 수정
sudo vi .env
### .env
DOMAIN_OR_PUBLIC_IP={server-domain.com}
OPENVIDU_SECRET={openvidu-password}
CERTIFICATE=letsencrypt
LETSENCRYPT_EMAIL={your-email.com}
OPENVIDU_RECORDING=true
# 해당 파일 저장 후 OpenVidu 실행
./openvidu start
# container 모두 실행된 후 -> openvidu media server SSL 적용완료
./openvidu stop

# 2-2. .env 파일 수정
sudo vi .env
HTTP_PORT={HTTP_PORT:8442}
HTTPS_PORT={HTTPS_PORT:8443}

# 3. OpenVidu 실행 ( 적용 완료 )
./openvidu start
```

2.5. Nginx 설치 및 설정

```
# 1. nginx 설치
cd ~
# sudo apt update
sudo apt install nginx

# 2. myapp.conf 작성
sudo vi /etc/nginx/conf/myapp.conf
# 아래 myapp.conf 내용 복사 후 붙여넣기

# nginx 시작
sudo systemctl start nginx
# 부팅 시 자동 실행 설정
sudo systemctl enable nginx
# nginx 상태 확인
sudo systemctl status nginx
```

```
# nginx 재시작
sudo systemctl restart nginx
```

▼ myapp.conf

```
# HTTP 서버 블록: 모든 HTTP 트래픽을 HTTPS로 리다이렉트
server {
    listen 80;
    server_name {server-domain.com};
    return 301 https://$server_name$request_uri;
}

# HTTPS 서버 블록
server {
    listen 443 ssl;
    server_name i11e201.p.ssafy.io;

    # SSL 인증서 설정
    ssl_certificate /etc/letsencrypt/live/i11e201.p.ssafy.io/fullch
ain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i11e201.p.ssafy.io/pr
ivkey.pem;

    # SSL 설정 강화
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA5
12:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-
AES256-SHA384;
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:50m;
    ssl_session_tickets off;
    ssl_stapling on;
    ssl_stapling_verify on;
    resolver 8.8.8.8 8.8.4.4 valid=300s;
    resolver_timeout 5s;

    # HSTS 설정
    add_header Strict-Transport-Security "max-age=63072000; include
Subdomains; preload";

    # 보안 헤더 설정
    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
```

```
    add_header X-Content-Type-Options "nosniff";

    # Jenkins 관련 설정
    location ^~ /jenkins {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host:$server_port;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_fo
r;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_redirect off;
        proxy_set_header X-Forwarded-Host $host:$server_port;
        proxy_set_header X-Forwarded-Prefix /jenkins;

        proxy_connect_timeout 150;
        proxy_send_timeout 100;
        proxy_read_timeout 100;

        proxy_buffer_size 8k;
        proxy_buffers 4 32k;
        proxy_busy_buffers_size 64k;
        proxy_temp_file_write_size 64k;
    }

    # Jenkins 루트 경로 리다이렉트
    location = /jenkins {
        return 301 $scheme://$host$request_uri/;
    }

    # API 서버 관련 설정
    location /api {
        proxy_pass http://localhost:8081;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_fo
r;
        proxy_set_header X-Forwarded-Proto $scheme;

        # CORS 설정
        add_header 'Access-Control-Allow-Origin' '*';
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIO
NS, PUT, DELETE';
        add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X
-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Rang
```

```
e';
    }

    # React 앱을 위한 location 블록
    location / {
        proxy_pass http://localhost:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_fo
r;
        proxy_set_header X-Forwarded-Proto $scheme;

        try_files $uri $uri/ /index.html;
    }

    # 정적 파일 캐싱 설정
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
        proxy_pass http://localhost:3000;
        proxy_cache_valid 200 60m;
        proxy_cache_use_stale error timeout http_500 http_502 http_
503 http_504;
        add_header Cache-Control "public, max-age=31536000, immutab
le";
        access_log off;
    }

    # gzip 압축 설정
    gzip on;
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_types text/plain text/css text/xml application/json applic
ation/javascript application/rss+xml application/atom+xml image/svg
+xml;
}
```

2.6. Whisper 설치

```
# 1. Python 및 torch 라이브러리 설치
sudo apt update -y
sudo apt install python3-pip
pip3 install torch

# 2. .profile 파일 수정
```

```
vi ~/.profile
### .profile
export PATH=$PATH:/home/ubuntu/.local/bin
# 해당 파일 저장 후 PATH 확인 (/home/ubuntu/.local/bin 있는지 확인)
sudo sh
echo $PATH

# 3. ffmpeg, openai-whisper 라이브러리 설치
sudo apt install ffmpeg
pip3 install openai-whisper

# 4. 설치 버전 확인
whisper --vesion
# 사용법
whisper --help
```

2.7. BERT Model 다운로드

```
# 1. 디렉토리 생성
cd ~
mkdir models
cd models

# 2. Git Clone
sudo apt-get install git-lfs #대용량 모델 Clone을 위한 git-lfs 설치
git lfs install  # Git LFS 초기화
git clone {model-url:letr-sol-profanity-filter}
cd {github-model-directory}
git lfs pull  # LFS 파일 다운로드
```

2.8. Jenkins 루트 폴더 생성 및 Jenkins.war 다운로드

```
# 1. jenkins_home 폴더 생성 및 이동
cd ~
mkdir jenkins_home
cd jenkins_home

# 2. 최신 버전 jenkins 다운로드
# 일부 젠킨스 미러 사이트(중국)가 접속되지 않는 현상이 발생하여
# 직접 jenkins.war 파일 다운로드 후 docker-compose로 볼륨 마운트를 수행
wget https://mirrors.cloud.tencent.com/jenkins/war/2.469/jenkins.war
sudo chown 1000:1000 ~/jenkins_home/jenkins.war
```

2.9. AWS S3 계정 연결

```
# 1. aswcli 설치
sudo apt-get update
sudo apt-get install awscli inotify-tools

# 2. aws configure 설정
aws configure
export AWS_ACCESS_KEY_ID={aws-access-key}
export AWS_SECRET_ACCESS_KEY={aws-secret-access-key}
export AWS_DEFAULT_REGION={aws-default-region}
```

3. Deploy

3.1. Frontend (Dockerfile)

```
# 1. Home Directory로 이동
cd ~

# 2. Git Clone
git clone {gitlab-project-domain.git}

# 3. Move to Working Directory
cd {gitlab-project-domain}/{frontend-project-root-directory}

# 4. Write Dockerfile
sudo vi Dockerfile

### Dockerfile
FROM node:20.15.0 as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html
EXPOSE 3000
```

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

3.2. Backend (Dockerfile)

```
# 1. Home Directory로 이동
cd ~

# 2. Git Clone (생략)
# git clone {gitlab-project-domain.git}

# 3. Move to Working Directory
cd {gitlab-project-domain}/{backend-project-root-directory}

# 4. Write Dockerfile
sudo vi Dockerfile

### Dockerfile
FROM gradle:7.2-jdk17 AS build
WORKDIR /app
COPY gradle/ gradle/
COPY gradlew .
COPY build.gradle settings.gradle ./
COPY . .
RUN chmod +x ./gradlew
RUN ./gradlew build -x test --no-daemon

FROM openjdk:17-jdk
WORKDIR /app
COPY --from=build /app/build/libs/{backend-project-name}-0.0.1-SNAPSHOT.j
EXPOSE 8081
ENV JAVA_TOOL_OPTIONS -Djava.net.preferIPv4Stack=true
ENTRYPOINT ["java", "-Duser.timezone=Asia/Seoul", "-jar", "app.jar"]
### ENDS
```

3.3. Docker-Compose (Frontend, Backend)

DockerHub 에 가입한 username을 <docker-hub-username> 에 기입합니다

```
# 1. Home Directory로 이동
cd ~

# 2. Git Clone (생략)
```

```
# git clone {gitlab-project-domain.git}

# 3. Move to Working Directory
cd {gitlab-project-domain}

# 4. Write Docker-compose file
sudo vi docker-compose.yml

### docker-compose.yml
version: '3'

services:
  spring-app:
    image: ${BACKEND_IMAGE_NAME:-<docker-hub-username>/backend-imag
e}:${BUILD_NUMBER:-latest}
    ports:
      - "8081:8081"
    environment:
      - SPRING_DATASOURCE_URL=jdbc:{image-name:mysql}://{project-domai
n}:{port-number:3306}/{schema-name}?useSSL=false&useUnicode=true&serve
rTimezone=Asia/Seoul&allowPublicKeyRetrieval=true
      - SPRING_DATASOURCE_USERNAME={mysql-username}
      - SPRING_DATASOURCE_PASSWORD={mysql-password}
      - SERVER_SERVLET_CONTEXT_PATH=/api

  react-app:
    image: ${FRONTEND_IMAGE_NAME:-<docker-hub-username>/frontend-imag
e}:${BUILD_NUMBER:-latest}
    ports:
      - "3000:80"
### ENDS
```

3.4. CI/CD (Jenkins, MySQL, Flask)

```
# 1. Home Directory로 이동
cd ~

# 2. Jenkins root directory 생성
mkdir jenkins_home

# 2. Git Clone (생략)
# git clone {gitlab-project-domain.git}
```

```
# 3. Move to Working Directory
cd {gitlab-project-domain}/{ci-cd-root-directory:cicd}
```

3.4.1. Jenkins Dockerfile 생성

```
sudo vi Dockerfile

### Dockerfile
FROM jenkins/jenkins:lts

USER root

# Docker CLI 설치
RUN apt-get update && \
    apt-get -y install apt-transport-https ca-certificates curl gnupg2
software-properties-common && \
    curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key
add - && \
    add-apt-repository "deb [arch=amd64] https://download.docker.com/l
inux/debian $(lsb_release -cs) stable" && \
    apt-get update && \
    apt-get -y install docker-ce-cli

# Docker Compose 설치
RUN curl -L "https://github.com/docker/compose/releases/download/1.29.
2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-com
pose && \
    chmod +x /usr/local/bin/docker-compose
### ENDS
```

3.4.2. Flask Server 파일 작성

```
# 1. app 폴더 생성 및 이동
mkdir app
cd app

# 2. 아래 3개 파일 작성
sudo vi {Dockerfile && app.py && requirements.txt}
```

▼ Dockerfile

```
FROM python:3.8-slim
```

```
WORKDIR /app

RUN pip install --upgrade pip

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app.py .

ENV PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION=python
ENV CUDA_VISIBLE_DEVICES=-1
ENV TF_FORCE_GPU_ALLOW_GROWTH=true

CMD ["python", "app.py"]
```

▼ app.py

```
from flask import Flask, request, jsonify
import torch
from transformers import BertTokenizer, BertForSequenceClassificati
on
from keras.preprocessing.sequence import pad_sequences
import torch.nn.functional as F
import numpy as np
import os

app = Flask(__name__)

# 전역 변수로 모델과 토크나이저 선언
model = None
tokenizer = None
device = None
category_map = {
    "0": "일반발언",
    "1": "공격발언",
    "2": "혐오발언"
}

def load_model():
    global model, tokenizer, device
    model_path = '/app/models/{model-url:letr-sol-profanity-filte
r}'
    model = BertForSequenceClassification.from_pretrained(model_pat
h)
    tokenizer = BertTokenizer.from_pretrained(model_path)
```

```
    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
    model.to(device)
    print('Model load Finished!')

# 입력 데이터 변환
def convert_input_data(sentences):
    tokenized_texts = [tokenizer.tokenize(sent) for sent in sentenc
es]
    MAX_LEN = 128
    input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokeni
zed_texts]
    input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="lon
g", truncating="post", padding="post")
    attention_masks = []

    for seq in input_ids:
        seq_mask = [float(i>0) for i in seq]
        attention_masks.append(seq_mask)

    inputs = torch.tensor(input_ids)
    masks = torch.tensor(attention_masks)

    return inputs, masks

def test_sentences(sentences):
    model.eval()
    inputs, masks = convert_input_data(sentences)
    b_input_ids = inputs.to(device)
    b_input_mask = masks.to(device)

    with torch.no_grad():
        outputs = model(b_input_ids,
                        token_type_ids=None,
                        attention_mask=b_input_mask)
    logits = outputs[0]
    logits = np.array(F.softmax(logits.detach().cpu()))
    category = np.argmax(logits)
    return {
        'normal': format(logits[0][0], ".4f"),
        'offensive': format(logits[0][1], ".4f"),
        'hate': format(logits[0][2],".4f"),
        'category': category_map[str(category)]
    }
```

```
# Flask API 작성
@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    sentence = data['sentence']
    result = test_sentences([sentence])
    return jsonify(result)

if __name__ == '__main__':
    load_model()  # 서버 시작 시 모델 로드
    app.run(host='0.0.0.0', port=5000)
```

▼ requirements.txt

```
Flask==2.0.2
Werkzeug==2.0.2
tensorflow==2.8.0
keras==2.8.0
tokenizers==0.10.3
torch==1.6.0
transformers==4.11.3
protobuf==3.20.0
```

3.4.3. Docker-compose.yml 작성 및 실행

⚠ 2.8. Jenkins 루트 폴더 생성 및 Jenkins.war 다운로드 과정을 진행해야 함

→ 볼륨 마운트 과정에서 파일 및 폴더가 존재하지 않으므로 에러 발생

```
# 1. ci/cd directory로 이동
cd ..
#또는 cd ~/{gitlab-project-domain}/{ci-cd-root-directory:cicd}

# 2. docker-compose.yml 작성
sudo vi docker-compose.yml

### docker-compose.yml
version: '3'

services:
  jenkins:
    build: .  # 커스텀 Dockerfile 사용
    user: root
    environment:
      - JENKINS_OPTS="--prefix=/jenkins"
```

```
      - JENKINS_ARGS="--prefix=/jenkins"
    ports:
      - "8080:8080"
    volumes:
      - /home/ubuntu/jenkins_home:/var/jenkins_home:z
      - /home/ubuntu/jenkins_home/jenkins.war:/usr/share/jenkins/jenkins
      - /var/run/docker.sock:/var/run/docker.sock

  mysql:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: {mysql-password}
      MYSQL_DATABASE: {schema-name}
    volumes:
      - mysql_data:/var/lib/mysql
    ports:
      - "3306:3306"

  python:
    build:
      context: ./app
      dockerfile: Dockerfile
    ports:
      - "8082:5000"
    volumes:
      - /home/ubuntu/models:/app/models:z

volumes:
  mysql_data:
    external: true
### ENDS


# 3. docker-compose 실행
sudo docker-compose up -d

# docker-compose 종료, 컨테이너 삭제
sudo docker-compose down
```

3.5 Shell Script

3.5.1. Openvidu-Whisper-S3 파이프라인 스크립트 작성

```
# 1. script 폴더 생성
cd ~
mkdir scripts
cd scripts
mkdir logs

# 2. openvidu-s3-upload.sh, openvidu-cleanup.sh 작성
sudo vi {openvidu-s3-upload.sh && openvidu-cleanup.sh}
```

▼ openvidu-s3-upload.sh

```
#!/bin/bash

RECORDING_DIR="/opt/openvidu/recordings"
S3_BUCKET={"bucket-name"}
LOG_DIR="/home/ubuntu/scripts/logs"
LOG_FILE="$LOG_DIR/openvidu-s3-upload.log"

# 시스템 리소스에 따라 조정 가능한 변수
MAX_CONCURRENT_PROCESSES=3
WHISPER_TIMEOUT=3600  # STT 처리 타임아웃 (초)

mkdir -p "$LOG_DIR"
touch "$LOG_FILE"
chmod 755 "$LOG_DIR"
chmod 644 "$LOG_FILE"

# 프로세스 ID를 저장할 배열 선언
declare -a PIDS

upload_file() {
    local FILE="$1"
    local RELATIVE_PATH="${FILE#$RECORDING_DIR/}"
    local S3_FILE_PATH="recordings/$RELATIVE_PATH"
    local FILE_SIZE=$(du -m "$FILE" | cut -f1)
    local CONTENT_TYPE=""

    # WEBM 파일이고 40MB 이상인 경우 업로드 중단
    if [[ "$FILE" == *.webm && $FILE_SIZE -ge 40 ]]; then
        echo "$(date): WEBM 파일 크기가 40MB 이상입니다. 업로드하지 않습니
다: $FILE ($FILE_SIZE MB)" >> "$LOG_FILE"
        return
    fi
```

```
    if aws s3 ls "s3://$S3_BUCKET/$S3_FILE_PATH" > /dev/null 2>&1;
then
        echo "$(date): 파일이 이미 S3에 존재합니다: $S3_FILE_PATH" >>
"$LOG_FILE"
        return
    fi

    # 파일 확장자에 따라 Content-Type 설정
    if [[ "$FILE" == *.txt ]]; then
        CONTENT_TYPE="text/plain; charset=utf-8"
    elif [[ "$FILE" == *.webm ]]; then
        CONTENT_TYPE="audio/webm"
    elif [[ "$FILE" == *.mp4 ]]; then
        CONTENT_TYPE="video/mp4"
    fi

    # Content-Type을 지정하여 S3에 업로드
    aws s3 cp "$FILE" "s3://$S3_BUCKET/$S3_FILE_PATH" --content-typ
e "$CONTENT_TYPE"

    if [ $? -eq 0 ]; then
        echo "$(date): 파일이 성공적으로 S3에 업로드되었습니다: $S3_FILE_PA
TH" >> "$LOG_FILE"
    else
        echo "$(date): S3 업로드 실패: $FILE" >> "$LOG_FILE"
    fi
}

process_and_upload_file() {
    local FILE="$1"
    local IS_NEW_FILE="$2"
    local PREV_SIZE=0
    local CURRENT_SIZE=0

    while true; do
        CURRENT_SIZE=$(du -b "$FILE" | cut -f1)
        if [ "$CURRENT_SIZE" == "$PREV_SIZE" ]; then
            echo "$(date): 파일 크기가 안정화되었습니다: $FILE" >> "$LOG_
FILE"
            upload_file "$FILE"

            # Whisper STT 처리 (webm 파일에 대해서만)
            if [[ "$FILE" == *.webm ]]; then
                local FILENAME=$(basename "$FILE")
                local DIRNAME=$(dirname "$FILE")
```

```
                local TXT_FILE="${DIRNAME}/${FILENAME%.webm}.txt"

                local FILE_SIZE_MB=$((CURRENT_SIZE / 1048576))

                # 기존 파일: 1MB 미만, 새 파일: 40MB 미만일 때 STT 수행
                if { [ "$IS_NEW_FILE" = "true" ] && [ $FILE_SIZE_MB
-lt 40 ]; } || \
                   { [ "$IS_NEW_FILE" = "false" ] && [ $FILE_SIZE_M
B -lt 1 ]; }; then
                    if [ ! -f "$TXT_FILE" ]; then
                        echo "$(date): Whisper STT 처리 시작: $FILE"
>> "$LOG_FILE"

                        # Whisper 실행 (출력 제어)
                        timeout $WHISPER_TIMEOUT whisper --language
=ko --output_dir="$DIRNAME" --output_format=txt "$FILE" > /dev/null
2>&1
                        if [ $? -eq 124 ]; then
                            echo "$(date): Whisper STT 처리 시간 초과:
$FILE" >> "$LOG_FILE"
                        elif [ -f "$TXT_FILE" ]; then
                            echo "$(date): Whisper STT 처리 완료, 텍스
트 파일 업로드 시작: $TXT_FILE" >> "$LOG_FILE"
                            upload_file "$TXT_FILE"
                        else
                            echo "$(date): Whisper STT 처리 실패 또는
텍스트 파일 없음: $TXT_FILE" >> "$LOG_FILE"
                        fi
                    else
                        echo "$(date): STT 처리 건너뜀 (이미 .txt 파일
존재): $FILE" >> "$LOG_FILE"
                    fi
                else
                    echo "$(date): STT 처리 건너뜀 (파일 크기 제한 초과):
$FILE" >> "$LOG_FILE"
                fi
            fi

            break
        fi
        PREV_SIZE=$CURRENT_SIZE
        sleep 10
    done
}
```

```
# 기존 파일 처리
echo "$(date): 기존 파일 처리 시작" >> "$LOG_FILE"
find "$RECORDING_DIR" -type f \( -name "*.mp4" -o -name "*.webm" -o
-name "*.txt" \) | while read FILE
do
    echo "$(date): 기존 파일 발견: $FILE" >> "$LOG_FILE"
    process_and_upload_file "$FILE" "false" &
    PIDS+=($!)

    # 동시 실행 프로세스 수 제한
    while [ ${#PIDS[@]} -ge $MAX_CONCURRENT_PROCESSES ]; do
        for i in "${!PIDS[@]}"; do
            if ! kill -0 ${PIDS[i]} 2>/dev/null; then
                unset PIDS[i]
            fi
        done
        PIDS=("${PIDS[@]}")  # 배열 재정렬
        sleep 1
    done
done

# 기존 파일 처리 완료 대기
while [ ${#PIDS[@]} -gt 0 ]; do
    for i in "${!PIDS[@]}"; do
        if ! kill -0 ${PIDS[i]} 2>/dev/null; then
            unset PIDS[i]
        fi
    done
    PIDS=("${PIDS[@]}")  # 배열 재정렬
    sleep 1
done

echo "$(date): 기존 파일 처리 완료" >> "$LOG_FILE"

# 새 파일 감시 시작
echo "$(date): 새 파일 감시 시작" >> "$LOG_FILE"

inotifywait -m -r -e close_write --format '%w%f' "$RECORDING_DIR" |
while read FILE
do
    if [[ "$FILE" == *.webm || "$FILE" == *.mp4 || "$FILE" == *.txt
]]; then
        echo "$(date): 새 파일 감지됨: $FILE" >> "$LOG_FILE"
        process_and_upload_file "$FILE" "true" &
        PIDS+=($!)
```

```
        # 동시 실행 프로세스 수 제한
        while [ ${#PIDS[@]} -ge $MAX_CONCURRENT_PROCESSES ]; do
            for i in "${!PIDS[@]}"; do
                if ! kill -0 ${PIDS[i]} 2>/dev/null; then
                    unset PIDS[i]
                fi
            done
            PIDS=("${PIDS[@]}")  # 배열 재정렬
            sleep 1
        done
    fi
done

# 스크립트 종료 전 모든 백그라운드 프로세스 완료 대기
for PID in "${PIDS[@]}"; do
    wait $PID
done
```

▼ openvidu-cleanup.sh

```
#!/bin/bash

RECORDING_DIR="/opt/openvidu/recordings"
LOG_DIR="/home/ubuntu/openvidu-logs"
LOG_FILE="$LOG_DIR/openvidu-cleanup.log"

# 로그 디렉토리 및 파일 생성 (sudo 권한 필요)
sudo mkdir -p "$LOG_DIR"
sudo touch "$LOG_FILE"
sudo chmod 755 "$LOG_DIR"
sudo chmod 644 "$LOG_FILE"

log_message() {
    echo "$(date): $1" | sudo tee -a "$LOG_FILE"
}

log_message "녹화 파일 삭제 작업 시작"

if [ ! -d "$RECORDING_DIR" ]; then
    log_message "오류 - $RECORDING_DIR 디렉토리가 존재하지 않습니다."
    exit 1
fi

# 모든 파일 및 디렉토리 삭제 (sudo 권한으로 실행)
```

```
    sudo find "$RECORDING_DIR" -mindepth 1 -delete

    if [ $? -eq 0 ]; then
        log_message "모든 녹화 파일 및 디렉토리가 성공적으로 삭제되었습니다."
    else
        log_message "오류 - 일부 파일 또는 디렉토리를 삭제하는 데 실패했습니다."

        # 삭제 실패한 항목 확인
        FAILED_ITEMS=$(sudo find "$RECORDING_DIR" -mindepth 1)
        if [ -n "$FAILED_ITEMS" ]; then
            log_message "삭제하지 못한 항목:"
            echo "$FAILED_ITEMS" | sudo tee -a "$LOG_FILE"
        fi
    fi

    log_message "녹화 파일 삭제 작업 완료"

    exit 0
```

3.5.2. 권한 설정 및 실행

```
# 1. 권한 설정
chmod +x ./openvidu-s3-upload.sh
chmod +x ./openvidu-cleanup.sh

sudo chown -R ubuntu:ubuntu /opt/openvidu/recordings
sudo chmod -R 755 /opt/openvidu/recordings
sudo chown -R ubuntu:ubuntu /home/ubuntu/openvidu-logs
sudo chmod -R 755 /home/ubuntu/openvidu-logs

# 2. 실행 및 종료
/bin/bash /home/ubuntu/scripts/openvidu-cleanup.sh

# 백그라운드에서 실행
nohup /bin/bash /home/ubuntu/scripts/openvidu-s3-upload.sh &

# 실행 중인지 확인하기
ps aux | grep openvidu-s3-upload.sh

# 실행 중인 프로세스 종료
pkill -f openvidu-s3-upload.sh
# 또는
killall -9 openvidu-s3-upload.sh
```

4. Jenkins Settings

4.1. Setting Accounts

- https://{server-domain.com}/jenkins 로 Jenkins에 접속합니다.
- 아래 절차에 따라 초기 계정을 생성합니다.

```
# 1. 초기 비밀번호 설정

# jenkins 컨테이너 ID 확인
docker ps | grep jenkins
# jenkins 컨테이너 접속
docker exec -it {컨테이너 ID} /bin/bash
# 초기 비밀번호 확인
cat /var/jenkins_home/secrets/initialAdminPassword

# "Unlock Jenkins" 페이지에서 찾은 비밀번호를 입력합니다.
# 플러그인 설치:
# "Install suggested plugins" 또는 필요한 플러그인만 선택합니다.
# 관리자 계정 생성:
# 사용자명, 비밀번호, 이메일 등을 입력하여 관리자 계정을 만듭니다.
# 설정 완료:
# "Start using Jenkins" 버튼을 클릭하여 설정을 마칩니다.
```

4.2. TroubleShootings for install plugins

일부 젠킨스 미러 사이트(중국)가 접속되지 않는 현상이 발생하여 플러그인 설치가 원활하지 않을 수 있습니다. 이에 대한 가이드라인을 안내해드립니다.

- 플러그인 설치에 이상이 없었다면 생략 가능

```
# 1. docker-compose 종료
cd /home/ubuntu/{gitlab-project-domain}/{ci-cd-root-directory:cicd}
sudo docker-compose down

# 2. jenkins_home 경로에서 configure 설정 변경
cd /home/ubuntu/jenkins_home
mkdir update-center-rootCAs
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/u
pdate-center.crt -O ./update-center-rootCAs/update-center.crt
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://r
aw.githubusercontent.com/lework/jenkins-update-center/master/updates/t
encent/update-center.json#' ./hudson.model.UpdateCenter.xml
```

```
# 3. docker-compose 재시작
cd /home/ubuntu/{gitlab-project-domain}/{ci-cd-root-directory:cicd}
sudo docker-compose up -d
```

4.3. Check SecurityRealm

```
sudo vi /home/ubuntu/jenkins_home/config.xml

### config.xml
<securityRealm class="hudson.security.HudsonPrivateSecurityRealm">
    <disableSignup>true</disableSignup>
</securityRealm>
### <disableSignup> 값이 true인지 확인하기
```

4.4. 플러그인 설치

[Jenkins 관리] > [Plugins] > [Available Plugins] 에 진입 후 아래 플러그인 설치

- GItLab
- GitLab API
- GitLab Branch Source
- Dokcer Pipeline

4.5. DockerHub, Gitlab 연동

4.5.1. Gitlab Access Token 생성

```
Gitlab 접속
1. 사용자 설정 > Access Tokens
 - 토큰 이름 입력, 만료일 설정
 - api, read_repository 스코프 선택
 - Create personal access token 클릭
2. 생성된 토큰을 안전한 곳에 저장
```

4.5.2. Jenkins - Gitlab 연동

```
Jenkins 접속
1. [Jenkins Dashboard] > [Jenkins 관리] > [Configure System] 클릭
2. GitLab 섹션 찾기
3. "GitLab connections" 추가 또는 기존 연결 수정
 - 연결 이름 입력 (예: "GitLab Server")
```

```
 - GitLab 서버 URL 입력 (예: https://{project-domain.com}/)
 - "Credentials" 드롭다운 메뉴에서 "Add" 클릭
   - "Jenkins" 선택
   - "Kind"에서 "GitLab API token" 선택
   - API 토큰 입력 (GitLab에서 생성한 Personal Access Token)
   - "ID"에 식별자 입력 (예: gitlab-api-token)
   - "설명"에 간단한 설명 추가
   - "Add" 버튼 클릭
  - "Test Connection" 버튼 클릭하여 연결 확인
```

4.5.3. DockerHub Personal Access Token 생성

```
DockerHub 접속 (https://hub.docker.com/)
1. [Account Settings] > [Personal access tokens] > [New access token]
2. 토큰 생성
 - Access Token Description: 토큰명 입력
 - Access Permissions: Read, Write, Delete 로 변경
3. "Generate" 입력 후 발급된 Access Token 저장
```

4.5.4. Jenkins - Docker Hub Credentials 추가

```
1. [Jenkins Dashboard] > [Jenkins 관리] > [Manage Credentials]
2. 'Stores scoped to Jenkins'에서 '(global)'을 클릭
3. 왼쪽 메뉴에서 'Add Credentials'를 클릭
4. 다음과 같이 필드 작성
   - Kind: Username with password
   - Scope: Global
   - Username: {docker-hub-username}
   - Password: {docker-hub-access-Token}
   - ID: docker-hub-credentials (원하는 이름 작성)
   - Description: Docker Hub Credentials
5. 'Create' 버튼을 클릭해서 저장
```

4.5.5. Jenkins Pipeline 설정

```
Jenkins 접속

1. [Jenkins Dashboard] > [새로운 Item] 클릭
  - 이름 입력 (예: "project-pipeline")
  - "Pipeline" 선택 후 "OK" 클릭
2. Pipeline 설정:
   - "설명" 필드에 간단한 설명 추가 (선택사항)
   - "Build Triggers" 섹션에서 "Build when a change is pushed to GitLab"
체크
```

```
   - "GitLab webhook URL" 기억해두기 (GitLab Webhook 설정에 사용)
   - "고급" 버튼 클릭
   - "Generate" 버튼을 클릭하여 Secret Token 생성
   - 생성된 토큰을 안전한 곳에 복사 (GitLab Webhook 설정에 사용)
3. Pipeline 스크립트 설정:
   - "Pipeline" 섹션에서 "Definition"이 "Pipeline script"인지 확인
   - Script에 아래 내용 작성

### Pipeline Scripts
pipeline {
    agent any

    environment {
        DOCKERHUB_CREDENTIALS = credentials('docker-hub-credentials')
        BACKEND_IMAGE_NAME = "{docker-hub-username}/backend-image"
        FRONTEND_IMAGE_NAME = "{docker-hub-username}/frontend-image"
    }

    stages {
        stage('Check Docker') {
            steps {
                script {
                    sh 'docker --version'
                    sh 'docker-compose --version'
                }
            }
        }

        stage('Clone Repository') {
            steps {
                git branch: 'develop',
                credentialsId: 'gitlab-credentials',
                url: "{gitlab-project-domain.git}"
            }
        }

        stage('Build and Push Images') {
            steps {
                script {
                    docker.withRegistry('', 'docker-hub-credentials')
{
                        dir('{backend-project-root-directory}') {
                            def backendImage = docker.build("${BACKEND
_IMAGE_NAME}:${BUILD_NUMBER}", "--no-cache .")
                            backendImage.push()
```

```
                        }
                        dir('{frontend-project-root-directory}') {
                            def frontendImage = docker.build("${FRONTE
ND_IMAGE_NAME}:${BUILD_NUMBER}", "--no-cache .")
                            frontendImage.push()
                        }
                    }
                }
            }
        }

        stage('Deploy with Docker Compose') {
            steps {
                script {
                    sh "sed -i 's|${BACKEND_IMAGE_NAME}:.*|${BACKEND_I
MAGE_NAME}:${BUILD_NUMBER}|g' docker-compose.yml"
                    sh "sed -i 's|${FRONTEND_IMAGE_NAME}:.*|${FRONTEND
_IMAGE_NAME}:${BUILD_NUMBER}|g' docker-compose.yml"

                    sh "docker-compose down"
                    sh "docker-compose up -d"
                }
            }
        }
        stage('Cleanup') {
            steps {
                script {
                    // 최신 5개 이미지만 유지하고 나머지 삭제
                    sh """
                        docker images | grep ${BACKEND_IMAGE_NAME} | s
ort -r | awk 'NR>5 {print \$3}' | xargs -r docker rmi
                        docker images | grep ${FRONTEND_IMAGE_NAME} |
sort -r | awk 'NR>5 {print \$3}' | xargs -r docker rmi
                    """
                }
            }
        }
    }

    post {
        always {
            sh "docker logout"
        }
        failure {
            echo 'Pipeline failed! Check the logs for details.'
```

```
        }
    }
}
### ENDS
```

4.5.6. Gitlab Webhook 설정

```
1. Gitlab 접속
2. [Settings] > [Webhooks]
  - "URL"에 Jenkins 파이프라인의 GitLab webhook URL 입력
  - "Name"에 원하는 이름 입력
  - "Secret Token" 에 Jenkins 파이프라인의 Secret Token 입력
  - "Trigger events"에서 "Push events" 선택
    - Wildcard Pattern 선택
    - 원하는 브랜치 명, 또는 Wildcard 패턴 입력
  - "SSL verification" 에 "Enable SSL verification" 체크
3. Add Webhook 클릭
4. 완료 후 [Test] > [Push events] 로 파이프라인 작동 여부 체크
```