포팅 매뉴얼 (1)

소유자	조 조성빈
태그	

1. 개발환경

1.1 Frontend

- react 18.3.1
    - axios 1.7.7
    - react-redux 9.1.2
    - react-router-dom 6.26.2
    - react-speech-recognition 3.10.0

    - vite 5.4.1
    - vite-plugin-pwa 0.20.5

1.2 Backend

- Java
    - JDK 17
    - Spring Boot 3.3.4
    - Gradle 8.10.2

1.3 Server

- Ubuntu 20.04 LTS
- Nginx 1.27.1
- Docker 27.2.1
- Jenkins 2.477

1.4 Spark Cluster

- Spark 3.5.3
- Python 3.8.10
- JDK 17

1.5 Database

- MySQL 9.0.1

1.6 UI/UX

- Figma

1.7 형상/이슈 관리

- Gitlab
- Jira

2. 환경변수

2.1 Backend

```
# DB
SPRING_DATASOURCE_URL
SPRING_DATASOURCE_USERNAME
SPRING_DATASOURCE_PASSWORD
SPRING_DATASOURCE_DRIVER-CLASS-NAME

# JWT
JWT_SECRET

# OpenAI API
OPENAI_API-KEY
```

3. EC2 세팅

3.1 도커 설치

```
# 1. 시스템 패키지 목록 업데이트
sudo apt update

# 2. Docker 엔진 설치
sudo apt install docker.io

# 3. Docker Compose 설치
sudo apt install docker-compose

# 4. Docker의 버전 확인
docker --version

# 5. 사용자가 Docker 명령어를 실행할 때 권한 문제를 피하기 위해 Docker 그룹에 사용자 추가
sudo usermod -aG docker $USER

# 6. 새 그룹의 권한을 적용하기 위해 현재 터미널 세션을 새로운 그룹으로 변경
sudo newgrp docker

# 7. Docker 서비스 재시작
sudo systemctl restart docker
```

3.2 Nginx 설치 및 세팅

```
# 1. nginx 도커 이미지 설치
docker pull nginx

# 2. certbot 설치
sudo apt-get install certbot

# 3. 인증서 발급
sudo certbot certonly --standalone -d j11b206.p.ssafy.io

# 4. 인증서가 저장된곳을 컨테이너 내부와 매핑
docker run -d \
  --name nginx \
  -p 80:80 \
  -p 443:443 \
  -v /etc/letsencrypt:/etc/letsencrypt:ro \
  nginx

# 5. nginx script 작성 (reverse proxy + SSL)
server {
    listen 443 ssl;
    server_name j11b206.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/j11b206.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j11b206.p.ssafy.io/privkey.pem;

    root /usr/share/nginx/html;
```

```
    index index.html;

    location / {
        proxy_pass http://j11b206.p.ssafy.io:5173;
        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://j11b206.p.ssafy.io:8080/api/;
        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 /swagger-ui/ {
        proxy_pass http://j11b206.p.ssafy.io:8080/swagger-ui/;
        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 /jenkins/ {
        proxy_pass http://j11b206.p.ssafy.io:9000/jenkins/;
        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;

        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-Port $server_port;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        proxy_redirect http://j11b206.p.ssafy.io:9000/ https://j11b206.p.ssafy.io/jenkins;
    }
}

server {
    listen 80;
    server_name j11b206.p.ssafy.io;
    return 301 https://$host$request_uri;
}

# 6. nginx 재실행
docker restart nginx
```

3.3 EC2 포트 정리

Port 번호	항목

22	SSH
80	HTTP
443	HTTPS
3306	MySQL (Docker)
8080	Spring Boot (Docker)
9000	Jenkins (Docker)

4. 빌드 및 배포

4.1 React

4.1.1 Dockerfile

```
FROM node:20.15-alpine AS build
WORKDIR /app
COPY package.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx:alpine
COPY nginx.conf /etc/nginx/nginx.conf
COPY --from=build /app/dist /usr/share/nginx/html
```

4.2 Spring Boot

4.2.1 Dockerfile

```
FROM openjdk:17-jdk-slim
WORKDIR /app
RUN apt-get update && apt-get install -y curl
COPY build/libs/log-service-0.0.1-SNAPSHOT.jar /app/log-service.jar
ENTRYPOINT ["java", "-jar", "/app/log-service.jar"]
```

4.2.2 docker-compose.yml

```
services:
  config-server:
    image: mrcsbin/helloword-config-server:latest
    container_name: config-server
    ports:
      - "8888:8888"
    networks:
      - backend-network
    environment:
      - SPRING_PROFILES_ACTIVE=docker,native
    healthcheck:
      test: [ "CMD", "curl", "-f", "http://config-server:8888/actuator/health" ]
      interval: 10s
      timeout: 5s
      retries: 5

  discovery-server:
    image: mrcsbin/helloword-discovery-server:latest
```

```
    container_name: discovery-server
    ports:
      - "8761:8761"
    depends_on:
      config-server:
        condition: service_healthy
    networks:
      - backend-network
    healthcheck:
      test: [ "CMD", "curl", "-f", "http://discovery-server:8761/actuator/health" ]
      interval: 10s
      timeout: 5s
      retries: 5
    environment:
      - SPRING_PROFILES_ACTIVE=docker

  api-gateway:
    image: mrcsbin/helloword-api-gateway:latest
    container_name: api-gateway
    ports:
      - "8080:8080"
    networks:
      - backend-network
    depends_on:
      discovery-server:
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=docker

  auth-service:
    image: mrcsbin/helloword-auth-service:latest
    container_name: auth-service
    networks:
      - backend-network
    depends_on:
      discovery-server:
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=docker

  collection-service:
    image: mrcsbin/helloword-collection-service:latest
    container_name: collection-service
    networks:
      - backend-network
    depends_on:
      discovery-server:
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=docker

  kid-service:
    image: mrcsbin/helloword-kid-service:latest
    container_name: kid-service
    networks:
      - backend-network
    depends_on:
      discovery-server:
```

```
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=docker

  probability-service:
    image: mrcsbin/helloword-probability-service:latest
    container_name: probability-service
    networks:
      - backend-network
    depends_on:
      discovery-server:
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=docker

  user-service:
    image: mrcsbin/helloword-user-service:latest
    container_name: user-service
    networks:
      - backend-network
    depends_on:
      discovery-server:
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=docker

  word-service:
    image: mrcsbin/helloword-word-service:latest
    container_name: word-service
    networks:
      - backend-network
    depends_on:
      discovery-server:
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=docker

  game-service:
    image: mrcsbin/helloword-game-service:latest
    container_name: game-service
    networks:
      - backend-network
    depends_on:
      discovery-server:
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=docker

  log-service:
    image: mrcsbin/helloword-log-service:latest
    container_name: log-service
    networks:
      - backend-network
    depends_on:
      discovery-server:
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=docker
```

```
networks:
  backend-network:
    driver: bridge
```

5. DB (MySQL)

```
# 1. MySQL Docker 이미지 다운로드
docker pull mysql

# 2. MySQL 컨테이너 실행 (컨테이너 생성)

# 2.1 auth-service-db 컨테이너 생성
docker run -d --name auth-service-db -e MYSQL_ROOT_PASSWORD=비밀번호 -p 13312:3306 mysql

# 2.2 probability-service-db 컨테이너 생성
docker run -d --name probability-service-db -e MYSQL_ROOT_PASSWORD=비밀번호 -p 13311:3306 mysql

# 2.3 collection-service-db 컨테이너 생성
docker run -d --name collection-service-db -e MYSQL_ROOT_PASSWORD=비밀번호 -p 13310:3306 mysql

# 2.4 word-service-db 컨테이너 생성
docker run -d --name word-service-db -e MYSQL_ROOT_PASSWORD=비밀번호 -p 13309:3306 mysql

# 2.5 kid-service-db 컨테이너 생성
docker run -d --name kid-service-db -e MYSQL_ROOT_PASSWORD=비밀번호 -p 13308:3306 mysql

# 2.6 log-service-db 컨테이너 생성
docker run -d --name log-service-db -e MYSQL_ROOT_PASSWORD=비밀번호 -p 13307:3306 mysql

# 2.7 unnamed-db 컨테이너 생성
docker run -d --name unnamed-db -e MYSQL_ROOT_PASSWORD=비밀번호 -p 13306:3306 mysql

# 3. MySQL 컨테이너에 접속 (예시: auth-service-db)
docker exec -it auth-service-db mysql -u root -p

# 4. MySQL에서 데이터베이스 생성
CREATE DATABASE daylog;

# 5. 생성된 데이터베이스 확인
SHOW DATABASES;
```

6. CI/CD 구축

6.1 Jenkins 설치

```
# 1. 젠킨스 이미지 다운로드 (docker)
docker pull jenkins/jenkins:jdk17

# 2. 젠킨스 컨테이너 실행
docker run -d \
  --name jenkins \
  -p 9000:8080 \
```

```
  -p 50000:50000 \
  -v jenkins_home:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -e JENKINS_OPTS="--prefix=/jenkins" \
  -e JENKINS_ARGS="--prefix=/jenkins" \
  -e JENKINS_HOME="/var/jenkins_home" \
  -e JAVA_OPTS="-Djenkins.install.runSetupWizard=false" \
  --network mynetwork \
  jenkins/jenkins:jdk17

# 3. 젠킨스 설정
http://ec2주소:9000 접속 후 계정과 기본 플러그인 설치
```

6.2 도커 명령어를 사용하기 한 CLI도 설치

```
docker exec -it --user root jenkins /bin/bash
apt-get update
apt-get install -y [docker.io](http://docker.io/)
```

6.3 Gitlab Credentials 발급

- gitlab의 Personal Access Token, Repository Access Token 발급 후 등록
- gitLab API Token 유형으로 등록 ID: 식별하기 좋은 이름 API Token: GitLab에서 발급받은 토큰
- gitlab Repo Token은 반드시 Username with Password로
- Username ⇒ Gitlab 이메일 (wlgnsdl12334@gmail.com)
- Password ⇒ Gitlab Repo Token
- ID ⇒ 식별하기 쉬운 이름

6.4 Dockar Hub Credentails 등록

- Username with password 유형으로 등록
- Username: 도커 허브 이메일(wlgnsdl12334@gmail.com)
- Password: 도커 허브 비밀번호 혹은 API 토큰

6.5 Gitlab과 Jenkins 연결

- Jenkins의 gitLab 부분에 가서 gitlab과 연결한다.
- Connection name ⇒ 아무거나 ⇒ daylog (프로젝트 이름)
- GitLab Host URL ⇒ 우리가 현재 작업하고 있는 도메인 사이트 ⇒ https://lab.ssafy.com
- Credentials ⇒ 아까 우리가 GitLab API Token을 등록한 Credentials 등록
- 후에 Test Connection으로 테스트. success라 뜨면 끝.

6.6 Jenkins에 SSH Key 등록

- Jenkins와 원격 서버 간의 SSH 연결을 하기 위하여 EC2 서버의 SSH 키를 Jenkins에 등록한다. 그러기 위해서는 SSH 키 쌍(비공개 키와 공개 키)을 생성한다. ssh-keygen #나오는 모든 입력값은 Enter, 그러면 비공개 키와 공개 키가 ~/.ssh에 생성된다
- **비공개 키 (Private Key)**: 일반적으로 `~/.ssh/id_rsa`라는 파일명으로 저장. 공유 X
- **공개 키 (Public Key)**: 일반적으로 `~/.ssh/id_rsa.pub`라는 파일명으로 저장. 공유 O
- 이제 Jenkins System에 들어가서 SSH키를 등록한다. 설치한 Plugin인 Publish Over SSH를 이용하여 EC2의 SSH키를 등록한다.
- key ⇒ .ssh/id_rsa(비공개 키)의 값. 밑에 꺼 —- 포함해서 다 복사해서 붙여야 함.

```
----BEGIN OPENSSH PRIVATE KEY------
대충 키 내용
-----END OPENSSH PRIVATE KEY---—
```

- Name ⇒ 식별하기 쉬운 이름(EC2 Server)
- HostName ⇒ DNS 혹은 Public ip (i11b107.p.ssafy.io)
- UserName ⇒ EC2에서 사용하는 Username (기본적으로는 ubuntu)
- 후에 Test Configuration 진행 후 Success면 끝

6.7 Jenkins에 Item(PipeLine) 생성

6.7.1 fe-develop

```
pipeline {
    agent any

    stages {
        stage('Checkout fe/develop') {
            steps {
                git(
                    branch: 'fe/develop',
                    credentialsId: 'git-login',
                    url: 'https://lab.ssafy.com/s11-bigdata-dist-sub1/S11P21B206.git'
                )
            }
        }

        stage('Install Dependencies') {
            steps {
                dir('frontend') {
                    sh 'npm install'
                }
            }
        }

        stage('Build') {
            steps {
                dir('frontend') {
                    sh 'npm run build'
                }
            }
        }

        stage('Push to fe/deploy (Rebase)') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'git-login', usernameVariable:
                    sh '''
                    REPO_URL="https://$(echo $GIT_USERNAME | sed 's/@/%40/g'):${GIT_PASSWORD}@la

                    git fetch $REPO_URL

                    git checkout fe/deploy || git checkout -b fe/deploy origin/fe/deploy

                    git rebase fe/develop

                    git push $REPO_URL fe/deploy --force
```

```
                    '''
                }
            }
        }
    }

    post {
        always {
            echo 'Rebase pipeline finished.'
        }
        success {
            echo 'Rebase successful, changes pushed to fe/deploy-test.'
        }
        failure {
            echo 'Rebase failed. Please resolve conflicts.'
        }
    }
}
```

6.7.2 fe-deploy

```
pipeline {
    agent any

    environment {
        DOCKER_CREDENTIALS = credentials('docker-hub-credentials')
        DOCKER_HUB_REPO = 'mrcsbin/helloword-frontend'
        IMAGE_TAG = 'latest'
    }

    stages {
        stage('Checkout fe/deploy') {
            steps {
                git(
                    branch: 'fe/deploy',
                    credentialsId: 'git-login',
                    url: 'https://lab.ssafy.com/s11-bigdata-dist-sub1/S11P21B206.git',
                    changelog: false,
                    poll: false
                )
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    dir('frontend') {
                        docker.withRegistry('https://registry.hub.docker.com', 'docker-hub-cr
edentials') {
                            // 캐시 사용 방지를 위해 --no-cache 옵션 추가
                            def image = docker.build("${DOCKER_HUB_REPO}:${env.IMAGE_TAG}",
"--no-cache .")
                            image.push()
                        }
                    }
                }
            }
```

```
        }

        stage('Deploy to EC2') {
            steps {
                sshagent(credentials: ['my-ssh-credentials']) {
                    withCredentials([string(credentialsId: 'EC2_SERVER_IP', variable: 'IP')])
{
                        sh """
                        ssh -o StrictHostKeyChecking=no ubuntu@$IP '
                        docker stop frontend || true
                        docker rm frontend || true
                        docker pull ${DOCKER_HUB_REPO}:${env.IMAGE_TAG}
                        docker run -d --name frontend -p 5173:80 ${DOCKER_HUB_REPO}:${env.IMA
GE_TAG}
                        '
                        """
                    }
                }
            }
        }

    post {
        always {
            echo 'fe/deploy pipeline finished.'
        }
        success {
            echo 'Frontend deployed successfully on EC2.'
        }
        failure {
            echo 'Frontend deployment failed.'
        }
    }
}
```

6.7.3 be-develop

```
pipeline {
    agent any

    stages {
        // 원격 브랜치 가져오기
        stage('Checkout be/develop') {
            steps {
                git(
                    branch: 'be/develop',
                    credentialsId: 'git-login',
                    url: 'https://lab.ssafy.com/s11-bigdata-dist-sub1/S11P21B206.git'
                )
            }
        }

        // 로컬 변경 사항을 무시하고 원격 브랜치로 덮어쓰는 단계 추가
        stage('Reset Local Changes') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'git-login', usernameVariabl
e: 'GIT_USERNAME', passwordVariable: 'GIT_PASSWORD')]) {
```

```
                    sh '''
                    REPO_URL="https://$(echo $GIT_USERNAME | sed 's/@/%40/g'):${GIT_PASSWORD}
@lab.ssafy.com/s11-bigdata-dist-sub1/S11P21B206.git"
                    git fetch $REPO_URL be/develop
                    git reset --hard origin/be/develop  # 로컬 변경 사항을 무시하고 원격 브랜치로 덮어
쓰기
                    '''
                }
            }
        }

        // 최신 커밋과 이전 커밋 사이의 변경 사항 감지
        stage('Detect Changed Services') {
            steps {
                script {
                    def changedFiles = sh(
                        script: 'git diff --name-only HEAD~1 HEAD -- backend/',
                        returnStdout: true
                    ).trim()

                    def changedServices = []

                    if (changedFiles) {
                        def fileList = changedFiles.split("\n")
                        for (file in fileList) {
                            if (file.startsWith("backend/")) {
                                def segments = file.split("/")
                                if (segments.length > 1) {
                                    def serviceDir = segments[1]
                                    if (!changedServices.contains(serviceDir) && serviceDir !
= ".idea" && serviceDir != "docker-compose.yml") {
                                        changedServices << serviceDir
                                    }
                                }
                            }
                        }
                    }

                    if (changedServices.size() > 0) {
                        env.CHANGED_SERVICES = changedServices.join(',')
                        echo "Changed services: ${env.CHANGED_SERVICES}"
                    } else {
                        echo "No changes detected in backend services."
                        env.CHANGED_SERVICES = ''
                    }
                }
            }
        }

        // 변경된 마이크로서비스에 대해 빌드 및 테스트
        stage('Build and Test Changed Services') {
            when {
                expression { return env.CHANGED_SERVICES != '' }
            }
            steps {
                script {
                    def servicesToBuild = env.CHANGED_SERVICES.split(',')
```

```
                    for (service in servicesToBuild) {
                        echo "Building and testing backend/${service}"

                        dir("backend/${service}") {
                            sh 'chmod +x ./gradlew || exit 0'
                            sh './gradlew clean build --build-cache || exit 0'
                            sh './gradlew test || exit 0'
                        }
                    }
                }
            }
        }

        // 리베이스 후 배포 브랜치에 푸시
        stage('Push to be/deploy') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'git-login', usernameVariabl
e: 'GIT_USERNAME', passwordVariable: 'GIT_PASSWORD')]) {
                    sh '''
                    REPO_URL="https://$(echo $GIT_USERNAME | sed 's/@/%40/g'):${GIT_PASSWORD}
@lab.ssafy.com/s11-bigdata-dist-sub1/S11P21B206.git"

                    # 원격 브랜치와 동기화
                    git fetch $REPO_URL
                    git reset --hard origin/be/develop  # 로컬 변경 사항 무시하고 원격 브랜치 상태로
덮어쓰기
                    git pull $REPO_URL be/develop  # 원격 브랜치의 최신 상태 확인

                    # 배포 브랜치로 체크아웃 후 리베이스
                    git checkout be/deploy || git checkout -b be/deploy origin/be-deploy
                    git rebase be/develop

                    # 강제 푸시
                    git push $REPO_URL be/deploy --force
                    '''
                }
            }
        }
    }

    post {
        always {
            echo 'Pipeline finished.'
        }
        success {
            echo 'Build and test successful.'
        }
        failure {
            echo 'Build or test failed.'
        }
    }
}
```

6.7.4 be-deploy

```
pipeline {
    agent any
```

```
    environment {
        IMAGE_TAG = 'latest'
    }

    parameters {
        booleanParam(name: 'CLEAN_WORKSPACE', defaultValue: true, description: 'Clean workspa
ce before build')
    }

    stages {
        stage('Clean Workspace') {
            steps {
                script {
                    if (params.CLEAN_WORKSPACE) {
                        echo 'Cleaning workspace...'
                        deleteDir()
                    } else {
                        echo 'Skipping workspace cleanup.'
                    }
                }
            }
        }

        stage('Checkout Source Code') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'git-login', usernameVariabl
e: 'GIT_USERNAME', passwordVariable: 'GIT_PASSWORD')]) {
                    script {
                        def encodedUsername = GIT_USERNAME.replaceAll("@", "%40")
                        def repoUrl = "https://${encodedUsername}:${GIT_PASSWORD}@lab.ssafy.c
om/s11-bigdata-dist-sub1/S11P21B206.git"

                        sh """
                        if [ -d .git ]; then
                          echo "Existing repository found, pulling changes..."
                          git pull origin be/deploy
                        else
                          echo "No existing repository found, cloning..."
                          git clone ${repoUrl} -b be/deploy .
                        fi
                        """
                    }
                }
            }
        }

        stage('Determine Changed Services') {
            steps {
                script {
                    def changedFiles = sh(
                        script: "git diff --name-only HEAD^ HEAD",
                        returnStdout: true
                    ).trim().split("\n")

                    def changedServices = changedFiles
                        .findAll { it.startsWith("backend/") }
                        .collect { it.split("/")[1] }
```

```
                    .unique()

                if (changedServices.isEmpty()) {
                    echo "No changes detected in backend services. Skipping build."
                    currentBuild.result = 'SUCCESS'
                    return
                }

                env.CHANGED_SERVICES = changedServices.join(",")
                echo "Changed services: ${env.CHANGED_SERVICES}"
            }
        }
    }

    stage('Build Backend Services') {
        steps {
            script {
                def services = env.CHANGED_SERVICES.split(",")
                services.each { service ->
                    dir("backend/${service}") {
                        echo "Building ${service} with Gradle"

                        // 권한 수정
                        sh 'chmod +x ./gradlew'

                        // Gradle 빌드
                        sh './gradlew clean build'
                    }
                }
            }
        }
    }


    stage('Build and Push Docker Images') {
        steps {
            script {
                def services = env.CHANGED_SERVICES.split(",")

                def parallelDockerBuilds = [:]
                services.each { service ->
                    service = service.replaceAll("@.*", "")

                    parallelDockerBuilds[service] = {
                        dir("backend/${service}") {
                            echo "Building Docker image for ${service}"

                            docker.withRegistry('https://registry.hub.docker.com', 'docke
r-hub-credentials') {
                                def image = docker.build("mrcsbin/helloword-${servic
e}:${env.IMAGE_TAG}", ".")
                                image.push()
                            }
                        }
                    }
                }
```

```
                    parallel parallelDockerBuilds
                }
            }
        }

        stage('Copy docker-compose.yml to EC2') {
            steps {
                sshagent(credentials: ['my-ssh-credentials']) {
                    withCredentials([string(credentialsId: 'EC2_SERVER_IP', variable: 'IP')])
{
                        script {
                            sh """
                            scp -o StrictHostKeyChecking=no backend/docker-compose.yml ubuntu
@${IP}:/home/ubuntu/docker-compose.yml
                            """
                        }
                    }
                }
            }
        }

        stage('Deploy to EC2 with Docker Compose') {
            steps {
                sshagent(credentials: ['my-ssh-credentials']) {
                    withCredentials([
                        string(credentialsId: 'EC2_SERVER_IP', variable: 'IP'),
                        usernamePassword(credentialsId: 'docker-hub-credentials', usernameVar
iable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')
                    ]) {
                        script {
                            sh """
                            ssh -o StrictHostKeyChecking=no ubuntu@${IP} << 'EOF'
                            echo '${DOCKER_PASSWORD}' | docker login -u '${DOCKER_USERNAME}'
--password-stdin

                            docker-compose -f /home/ubuntu/docker-compose.yml pull
                            docker-compose -f /home/ubuntu/docker-compose.yml up -d

                            exit
                            EOF
                            """
                        }
                    }
                }
            }
        }
    }

    post {
        always {
            echo 'Pipeline finished.'
        }
        success {
            echo 'Services successfully built and deployed.'
        }
        failure {
            echo 'Pipeline failed. Please check the logs.'
        }
```

```
    }
}
```

7. Spark Cluster 구성

7.1 JDK 설치

```
sudo apt update
sudo apt install openjdk-17-jdk -y
java -version
```

7.2 Python 설치

```
sudo apt install python3 -y
python3 --version
```

7.3 Apache Spark 설치

```
# spark 다운로드
wget https://downloads.apache.org/spark/spark-3.5.3/spark-3.5.3-bin-hadoop3.tgz

# 다운로드한 파일을 압축 해제
tar -xvf spark-3.5.3-bin-hadoop3.tgz
sudo mv spark-3.5.3-bin-hadoop3 /opt/spark

# Spark를 PATH에 추가
echo "export SPARK_HOME=/opt/spark" >> ~/.bashrc
echo "export PATH=$SPARK_HOME/bin:$PATH" >> ~/.bashrc
source ~/.bashrc
```

7.4 Spark 설정(standalone)

```
# 스파크 설정 디렉토리로 이동
cd /opt/spark/conf

# 템플릿 설정 파일 복사
cp spark-env.sh.template spark-env.sh
cp slaves.template workers

# spark-env.sh 파일을 열어 마스터와 워커를 설정
nano spark-env.sh
# 아래와 같이 추가
export SPARK_MASTER_HOST='localhost'
export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64

# workers 파일을 수정하여 워커 노드를 추가 (서버가 하나이므로 localhost로 설정)
nano workers
localhost

# 스파크 마스터를 시작
/opt/spark/sbin/start-master.sh
```

```
# 같은 서버에서 스파크 워커를 시작
/opt/spark/sbin/start-worker.sh spark://localhost:7077
```

7.5 Spark Code

```
from pyspark.sql import SparkSession
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.window import Window
from pyspark import StorageLevel
import findspark

# 스파크 세션 생성
findspark.init()
spark = SparkSession \
        .builder \
        .config("spark.driver.extraClassPath", "/home/hadoop/project/mysql-connector-j-8.0.33.ja
        .appName('pyspark-db-connect') \
        .getOrCreate()

# 로그 레벨 정의
spark.sparkContext.setLogLevel("ERROR")

# mysql 연결
ip = "j11b206.p.ssafy.io"
port_collection = "13310"
port_probability = "13311"
user = "root"
passwd = "root"
db = "helloword"
table_name = "collections"
table_answer_word_logs = "answer_word_logs"

df = spark.read.format("jdbc") \
                    .option("url", f"jdbc:mysql://{ip}:{port_collection}/{db}") \
                    .option("driver", "com.mysql.cj.jdbc.Driver") \
                    .option("dbtable", table_name) \
                    .option("user", user) \
                    .option("password", passwd) \
                    .load()

# TF 계산: 특정 kid_id와 word_id에 대한 answer_count / kid_id 그룹의 전체 answer_count 합 + 1
window_kid = Window.partitionBy("kid_id")
df_tf = df.withColumn("tf", F.col("count") / (F.sum("count").over(window_kid) + 1))

# IDF 계산: 전체 kid_id의 수 / word_id가 해당하는 answer_count가 1 이상인 kid_id의 수 + 1
total_kids = df.select("kid_id").distinct().count()

df_idf = df.groupBy("word_id") \
    .agg(F.countDistinct(F.when(F.col("count") >= 1, F.col("kid_id"))).alias("kid_count_for_word
    .withColumn("kid_count_for_word", F.coalesce(F.col("kid_count_for_word"), F.lit(0))) \
    .withColumn("idf", F.when(F.log(F.lit(total_kids) / (F.col("kid_count_for_word") + 1)) < 0,
                .otherwise(F.log(F.lit(total_kids) / (F.col("kid_count_for_word") + 1))))

# TF-IDF 계산
df_tfidf = df_tf.join(df_idf, "word_id", "left") \
    .withColumn("tfidf", F.round(F.col("tf") * F.col("idf"), 6))
```

```
# 1. 확률 계산: tfidf를 반전하여 등장 확률 계산
df_tfidf = df_tfidf.withColumn("inverse_tfidf", 1 / (F.col("tfidf") + 0.001))  # tfidf가 높을수록

# 2. kid_id 그룹별 확률 정규화
df_tfidf = df_tfidf.withColumn("prob_sum", F.sum("inverse_tfidf").over(window_kid))
df_tfidf = df_tfidf.withColumn("probability", F.col("inverse_tfidf") / F.col("prob_sum"))

# 3. 확률 반올림 (6번째 소수점 자리)
df_tfidf = df_tfidf.withColumn("probability", F.round(F.col("probability"), 6))

df_answer_logs = spark.read.format("jdbc") \
    .option("url", f"jdbc:mysql://{ip}:{port_probability}/{db}") \
    .option("driver", "com.mysql.cj.jdbc.Driver") \
    .option("dbtable", table_answer_word_logs) \
    .option("user", user) \
    .option("password", passwd) \
    .load()

# # 4. 결과 테이블 업데이트
df_updated = df_answer_logs.join(df_tfidf.select("id", "tfidf", "probability"), on="id", how="le
    .drop(df_answer_logs["probability"]) \
    .withColumnRenamed("probability", "probability") \
    .select("id", "kid_id", "word_id", "probability") \
    .orderBy("id")

# 데이터프레임 분산 저장을 통한 캐싱
df_updated.persist(StorageLevel.MEMORY_AND_DISK)

# 캐싱 완료를 위한 action
df_updated.count()

# mysql db에 데이터 전송
df_updated.write \
    .mode("overwrite") \
    .format("jdbc") \
    .option("truncate", "true") \
    .option("driver", "com.mysql.cj.jdbc.Driver") \
    .option("url", f"jdbc:mysql://{ip}:{port_probability}/{db}") \
    .option("dbtable", table_answer_word_logs) \
    .option("user", user) \
    .option("password", passwd) \
    .save()

# 데이터프레임 캐시 해제
df_updated.unpersist()

print("answer_word_logs 테이블의 tf-idf, 확률 계산 완료!")

# 스파크 세션 종료
spark.stop()
```