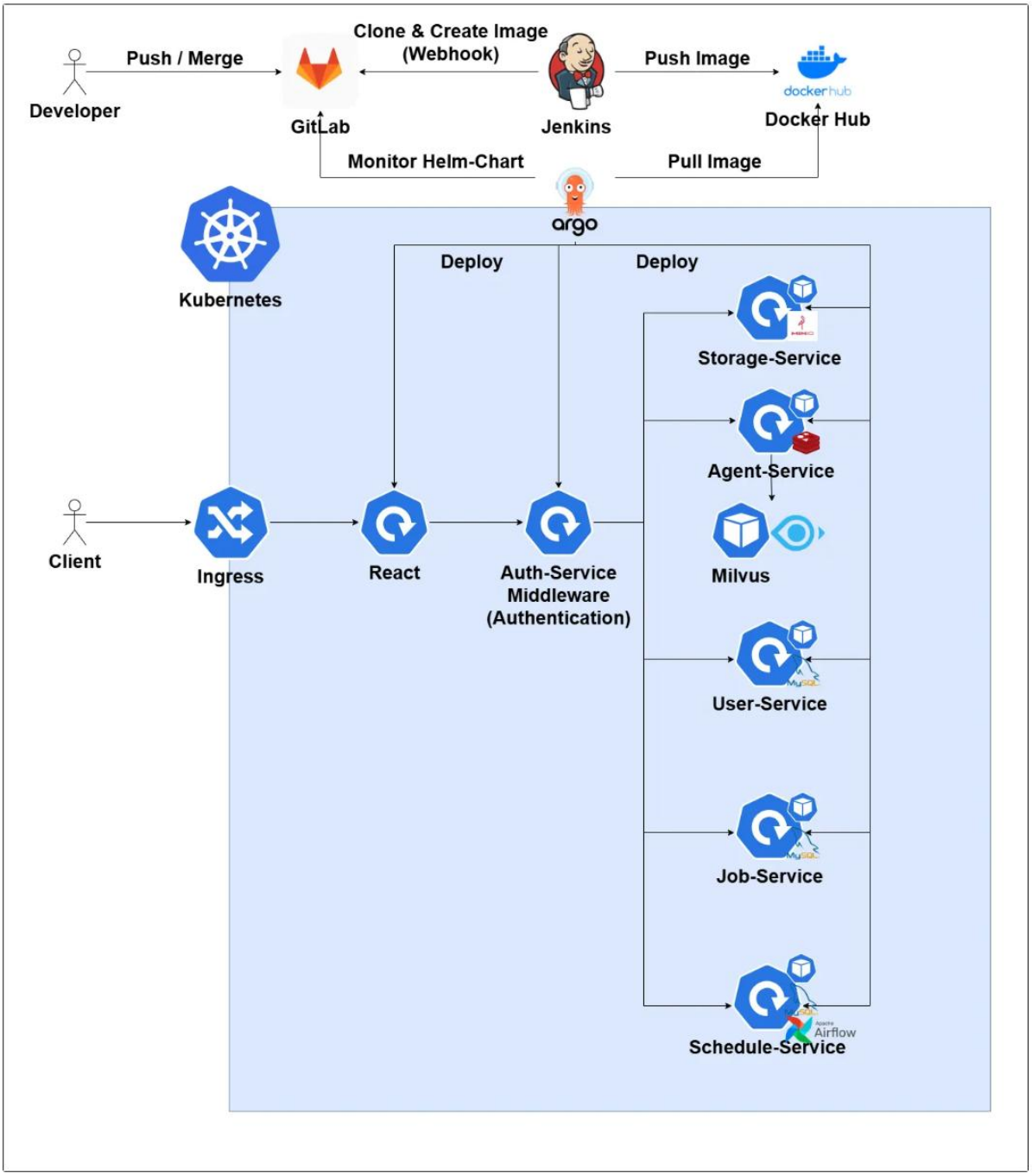
배포 검증 및 운영 가이드

내용

1. 시스템 개요

이 문서는 마이크로서비스 아키텍처를 배포하는 전체 과정을 설명합니다. K3s 기반 쿠버네티스 환경(v1.32.3+k3s1)과 Docker 를 사용할 수 없는 온프레미스 환경 모두에 대한 가이드를 제공합니다.

시스템 아키텍처



1. 개발자가 GitLab 에 코드를 Push/Merge
2. Jenkins 가 GitLab 웹훅을 통해 트리거되어 이미지 빌드 및 Docker Hub 에 Push
3. ArgoCD 가 GitLab 의 Helm 차트를 모니터링하고 Kubernetes 에 배포
4. 클라이언트가 Ingress 를 통해 애플리케이션에 접근하며, 인증 및 다양한 마이크로서비스로 라우팅

주요 구성 요소

CI/CD 파이프라인:

- GitLab: 소스 코드 관리 및 버전 제어
- Jenkins: 빌드 자동화 및 Docker 이미지 생성
- Docker Hub: 컨테이너 이미지 저장소
- ArgoCD: GitOps 기반 배포 자동화 도구

인프라 구성 요소:

- Kubernetes (K3s): 컨테이너 오케스트레이션 플랫폼
- Traefik: 인그레스 컨트롤러 및 로드 밸런서
- Milvus: 벡터 데이터베이스 (검색/추천 시스템용)
- MinIO: 객체 스토리지 시스템
- Airflow: 워크플로우 자동화 및 스케줄링 플랫폼

마이크로서비스:

- React: 프론트엔드 애플리케이션
- Auth Service: 인증 및 권한 관리 미들웨어
- Storage Service: 파일 저장 및 관리 (MinIO 사용)
- Agent Service: 에이전트 관리 및 제어 (Redis 사용, Milvus 와 연동)
- User Service: 사용자 프로필 및 정보 관리 (MySQL 사용)

- Job Service: 작업 정의 및 실행 관리 (MySQL 사용)
- Schedule Service: 작업 스케줄링 (Airflow 연동)

외부 접근 가능 서비스 포트:

- argocd-server (ArgoCD UI): NodePort 로 40157(HTTP) 및 30139(HTTPS) 포트 노출
- airflow-webserver (Airflow UI): NodePort 로 40159 포트 노출
- traefik (인그레스 컨트롤러): LoadBalancer 타입으로 외부 IP(172.26.10.158)에 30675(HTTP) 및 31642(HTTPS) 포트 노출

---

2. 마이크로서비스 아키텍처

현재 시스템은 마이크로서비스 아키텍처로 설계되어 있으며, 각 서비스는 독립적으로 개발, 배포 및 확장이 가능합니다. 아래는 마이크로서비스 간의 상호작용과 주요 역할을 설명합니다:

서비스 의존성 매트릭스

서비스	의존 대상 서비스
agent-service	auth-service, storage-service
auth-service	user-service
job-service	auth-service, agent-service
schedule-service	job-service, auth-service
storage-service	auth-service
user-service	auth-service
react (frontend)	모든 백엔드 서비스

서비스 흐름 및 상호작용

사용자 요청 경로:

- 외부 클라이언트 → Ingress(Traefik) → React(프론트엔드) → Auth-Service(인증) → 각 백엔드 서비스

인증 및 권한 관리:

- Auth-Service 는 미들웨어 역할을 하며 모든 요청에 대한 인증 및 권한 검증을 처리합니다.
- JWT 토큰 기반 인증을 사용하여 보안을 유지합니다.

서비스별 데이터 저장:

- 각 서비스는 자체 데이터베이스를 가지며, "데이터베이스 per 서비스" 패턴을 따릅니다.
- 서비스 간 직접적인 데이터베이스 접근은 허용되지 않으며, API 를 통해서만 통신합니다.

특수 서비스 역할:

- Storage-Service: MinIO 를 활용한 파일 저장 및 관리 서비스
- Agent-Service: Milvus 벡터 데이터베이스를 활용한 에이전트 관리
- Schedule-Service: Airflow 를 활용한 작업 자동화 및 스케줄링

서비스별 주요 기능 및 기술 스택

Agent-Service:

- 주요 기능: 에이전트 관리, 상태 모니터링, 명령 전달
- 기술 스택: FastAPI, Redis(캐싱), Milvus(벡터 검색)
- 의존 서비스: Auth-Service(인증), Storage-Service(파일 저장)

Auth-Service:

- 주요 기능: 사용자 인증, 토큰 관리, 권한 제어
- 기술 스택: FastAPI, JWT
- 의존 서비스: User-Service(사용자 정보)

User-Service:

- 주요 기능: 사용자 프로필 관리, 계정 설정
- 기술 스택: Spring Boot, MySQL
- 의존 서비스: Auth-Service(인증)

Job-Service:

- 주요 기능: 작업 정의, 실행, 결과 관리
- 기술 스택: FastAPI, MySQL
- 의존 서비스: Auth-Service(인증), Agent-Service(작업 실행)

Schedule-Service:

- 주요 기능: 작업 스케줄링, 워크플로우 관리
- 기술 스택: FastAPI, Airflow
- 의존 서비스: Job-Service(작업 실행), Auth-Service(인증)

Storage-Service:

- 주요 기능: 파일 업로드/다운로드, 메타데이터 관리
- 기술 스택: FastAPI, MinIO
- 의존 서비스: Auth-Service(인증)

React (프론트엔드):

- 주요 기능: 사용자 인터페이스, API 통합
- 기술 스택: React.js, Typescript
- 의존 서비스: 모든 백엔드 서비스

---

3. 사전 요구사항

하드웨어 요구사항

- 최소 4 CPU 코어
- 최소 16GB RAM
- 최소 100GB 스토리지

소프트웨어 요구사항

쿠버네티스 환경:

- Ubuntu 20.04 LTS 이상
- Docker 20.10 이상

온프레미스 환경 (Docker 없는 환경):

- Ubuntu 20.04 LTS 또는 다른 Linux 배포판
- Python 3.9 이상 (FastAPI 서비스)
- Java 17 이상 (Spring Boot 서비스)
- MySQL 8.0 이상
- Redis 6.0 이상

Git 저장소 구조

현재 프로젝트 구조는 다음과 같습니다:

```
S12P31S101/
├── backend/
│   ├── .idea/
│   ├── agent-service/
│   ├── auth-service/
│   ├── job-service/
│   ├── schedule-service/
│   ├── storage-service/
│   ├── user-service/
```

```
│   └── __init__.py
├── frontend/
└── helm-chart/
    ├── agent-service/
    ├── auth-service/
    ├── job-service/
    ├── schedule-service/
    │   ├── templates/
    │   │   ├── deployment.yaml
    │   │   ├── ingress.yaml
    │   │   ├── schedule-dags-pvc.yaml
    │   │   └── service.yaml
    │   ├── Chart.yaml
    │   └── values.yaml
    ├── storage-service/
    ├── user-service/
    └── .gitignore
├── README.md
├── Jenkinsfile
└── .gitignore
```

이 구조는 코드와 배포 구성을 명확히 분리합니다:

- backend/: 각 마이크로서비스의 독립적인 소스 코드를 포함
- frontend/: React 프론트엔드 애플리케이션 코드
- helm-chart/: 각 서비스별 Kubernetes 배포 구성 파일

---

4. 배포 환경별 설치 가이드

4.1. 쿠버네티스 기반 설치 가이드 (K3s)

K3s 설치

K3s 는 경량화된 쿠버네티스 배포판으로, 단일 바이너리로 제공됩니다.
현재 시스템에서는 v1.32.3+k3s1 버전을 사용합니다.

마스터 노드 설치

```
# K3s 설치 스크립트 실행 (특정 버전 지정)
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION=v1.32.3+k3s1 sh -

# 설치 확인
sudo kubectl get nodes

# kubeconfig 파일 접근 권한 설정
sudo chmod 644 /etc/rancher/k3s/k3s.yaml
mkdir -p ~/.kube
sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
sudo chown $(id -u):$(id -g) ~/.kube/config
export KUBECONFIG=~/.kube/config
```

워커 노드 추가 (필요시)

마스터 노드에서 토큰 확인:

```
sudo cat /var/lib/rancher/k3s/server/node-token
```

워커 노드에서 실행:

```
curl -sfL https://get.k3s.io | K3S_URL=https://마스터노드 IP:6443
K3S_TOKEN=토큰값 INSTALL_K3S_VERSION=v1.32.3+k3s1 sh -
```

필수 도구 설치

kubectl

```
curl -LO "https://dl.k8s.io/release/v1.32.3/bin/linux/amd64/kubectl"
chmod +x kubectl
```

```
sudo mv kubectl /usr/local/bin/
```

Helm

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
```

k9s (선택사항 - 관리용 UI 도구)

```
curl -sS https://webinstall.dev/k9s | bash
```

네임스페이스 생성

시스템의 각 구성 요소는 별도의 네임스페이스에 배포됩니다:

```
# 기본 네임스페이스 생성
kubectl create namespace argocd
kubectl create namespace agent-service
kubectl create namespace auth-service
kubectl create namespace job-service
kubectl create namespace schedule-service
kubectl create namespace storage-service
kubectl create namespace user-service
kubectl create namespace milvus
```

5. 서비스 배포

5.1. 인프라 서비스 배포

쿠버네티스 환경 (Helm 사용)

Traefik 배포

```
# Traefik CRD 설치
helm repo add traefik https://traefik.github.io/charts
helm repo update
helm install traefik-crd traefik/traefik-crd -n kube-system

# Traefik 설치
helm install traefik traefik/traefik -n kube-system
```

ArgoCD 설치 및 설정

```
# ArgoCD 설치
helm repo add argo https://argoproj.github.io/argo-helm
helm repo update
helm install argocd argo/argo-cd --namespace argocd --version 5.46.0

# ArgoCD CLI 설치
curl -sSL -o argocd-linux-amd64 https://github.com/argoproj/argo-
cd/releases/latest/download/argocd-linux-amd64
chmod +x argocd-linux-amd64
sudo mv argocd-linux-amd64 /usr/local/bin/argocd

# ArgoCD 초기 패스워드 확인
kubectl -n argocd get secret argocd-initial-admin-secret -o
jsonpath="{.data.password}" | base64 -d

# ArgoCD Image Updater 설치
helm install argocd-image-updater argo/argocd-image-updater --
namespace argocd --version 0.9.1
```

Milvus 배포

```
helm repo add milvus https://milvus-io.github.io/milvus-helm/
helm repo update

helm install milvus milvus/milvus \
```

```
  --namespace milvus \
  --set cluster.enabled=false \
  --set etcd.replicaCount=1 \
  --set minio.mode=distributed \
  --set minio.persistence.size=100Gi \
  --set minio.resources.requests.memory=1Gi \
  --set pulsar.enabled=false
```

MinIO 배포

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update

helm install minio bitnami/minio \
  --namespace storage-service \
  --set auth.rootUser=admin \
  --set auth.rootPassword=YOUR_MINIO_PASSWORD \
  --set persistence.size=50Gi
```

Airflow 배포

```
helm repo add apache-airflow https://airflow.apache.org
helm repo update

helm install airflow apache-airflow/airflow \
  --namespace schedule-service \
  --set executor=CeleryExecutor \
  --set postgresql.enabled=true \
  --set redis.enabled=true \
  --set webserver.defaultUser.password=YOUR_AIRFLOW_PASSWORD

# 배포된 Airflow 서버에 접근해 필요한 package 설치
kubectl exec -it airflow-scheduler-0 --namespace schedule-service --
/bin/bash
pip install minio pydantic-settings openpyxl

# utils 폴더를 airflow 내 폴더로 복사
kubectl cp ./exec/utils schedule-service/airflow-scheduler-
0:/opt/airflow/dags/utils
```

5.2. 마이크로서비스 배포

쿠버네티스 환경 (ArgoCD 사용)

ArgoCD 에 Git 저장소 등록

```
# ArgoCD 로그인
argocd login localhost:8080

# 저장소 등록
argocd repo add https://lab.ssafy.com/s12-final/S12P31S101.git --
username git-username --password git-password
```

마이크로서비스 배포

```
# 각 서비스 애플리케이션 생성
argocd app create agent-service \
  --repo https://lab.ssafy.com/s12-final/S12P31S101.git \
  --path helm-chart/agent-service \
  --dest-server https://kubernetes.default.svc \
  --dest-namespace agent-service \
  --sync-policy automated \
  --auto-prune \
  --self-heal \
  --revision be

# 나머지 서비스도 유사하게 생성
```

5.3 프론트엔드 애플리케이션 배포

쿠버네티스 환경 (ArgoCD 사용)

React 프론트엔드 배포

프론트엔드 React 애플리케이션도 ArgoCD 를 통해 배포합니다:

```
# ArgoCD 로 프론트엔드 애플리케이션 배포
argocd app create react \
  --repo https://lab.ssafy.com/s12-final/S12P31S101.git \
  --path helm-chart/react \
  --dest-server https://kubernetes.default.svc \
  --dest-namespace frontend \
  --sync-policy automated \
  --auto-prune \
  --self-heal
```

프론트엔드 Helm 차트 구조

```
helm-chart/react/
├── templates/
│   ├── deployment.yaml
│   ├── service.yaml
│   └── ingress.yaml
├── Chart.yaml
└── values.yaml
```

values.yaml 예시:

```yaml
replicaCount: 1

image:
  repository: k12s101ss/react
  tag: latest
  pullPolicy: Always

service:
  type: ClusterIP
  port: 80

ingress:
  enabled: true
  className: "traefik"
  annotations: {}
  hosts:
    - host: ""
      paths:
        - path: /
          pathType: Prefix
```

Ingress 를 통한 프론트엔드 라우팅

프로젝트에서는 Traefik Ingress 를 통해 프론트엔드를 노출합니다. 아래는 프론트엔드를 위한 Ingress 구성 예시입니다:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ include "react.fullname" . }}
  labels:
    {{- include "react.labels" . | nindent 4 }}
  {{- with .Values.ingress.annotations }}
  annotations:
```

```
    {{- toYaml . | nindent 4 }}
  {{- end }}
spec:
  ingressClassName: {{ .Values.ingress.className }}
  rules:
  {{- range .Values.ingress.hosts }}
    - host: {{ .host | quote }}
      http:
        paths:
        {{- range .paths }}
          - path: {{ .path }}
            pathType: {{ .pathType }}
            backend:
              service:
                name: {{ include "react.fullname" $ }}
                port:
                  number: {{ $.Values.service.port }}
        {{- end }}
  {{- end }}
```

프론트엔드 환경 변수 관리

프론트엔드(React) 애플리케이션의 환경 변수는 백엔드 서비스와 다른 방식으로 관리됩니다. React 애플리케이션은 빌드 시점에 환경 변수가 번들에 포함되므로, 배포 환경에 따라 다른 접근 방법이 필요합니다.

쿠버네티스 환경에서의 프론트엔드 환경 변수 관리

쿠버네티스 환경에서는, 다음과 같은 방법으로 프론트엔드 환경 변수를 관리하고 있습니다:

1. Jenkins CI/CD 파이프라인을 통한 환경 변수 주입

Jenkins 는 빌드 시점에 환경 변수를 프론트엔드 애플리케이션에 주입합니다:

```
// Jenkinsfile 예시
pipeline {
  agent any

  environment {
    // 전역 환경 변수 설정
    REACT_APP_API_BASE_URL = "http://172.26.10.158:30675"
  }

  stages {
    // ... 다른 스테이지들 ...

    stage('Build Frontend') {
      when {
        expression { return
env.CHANGED_TARGETS.contains('frontend') }
      }
      steps {
        dir('frontend') {
          // 환경 변수 파일 동적 생성
          sh """
            echo "REACT_APP_API_URL=${REACT_APP_API_BASE_URL}" > .env
            echo "REACT_APP_AUTH_SERVICE_PATH=/auth-service" >> .env
            echo "REACT_APP_USER_SERVICE_PATH=/user-service" >> .env
```

```
            echo "REACT_APP_AGENT_SERVICE_PATH=/agent-service"
>> .env
            echo "REACT_APP_JOB_SERVICE_PATH=/job-service" >> .env
            echo "REACT_APP_SCHEDULE_SERVICE_PATH=/schedule-service"
>> .env
            echo "REACT_APP_STORAGE_SERVICE_PATH=/storage-service"
>> .env
          """

          // 빌드 명령 실행
          sh 'npm ci'
          sh 'npm run build'

          // Docker 이미지 빌드 및 푸시
          sh """
            docker build -t k12s101ss/react:${GIT_COMMIT_SHORT}-
${BUILD_NUMBER} -t k12s101ss/react:latest .
            docker push k12s101ss/react:${GIT_COMMIT_SHORT}-
${BUILD_NUMBER}
            docker push k12s101ss/react:latest
          """
        }
      }
    }
  }
}
```

2. 배포 구성에서의 환경 변수 관리

Helm 차트의 values.yaml 파일에 환경 변수 설정을 포함하고, 이를 Deployment 매니페스트에 적용할 수 있습니다:

```yaml
# helm-chart/react/values.yaml
image:
  repository: k12s101ss/react
  tag: latest
  pullPolicy: Always

env:
  REACT_APP_API_URL: "http://172.26.10.158:30675"
  REACT_APP_AUTH_SERVICE_PATH: "/auth-service"
  REACT_APP_USER_SERVICE_PATH: "/user-service"
  REACT_APP_AGENT_SERVICE_PATH: "/agent-service"
  REACT_APP_JOB_SERVICE_PATH: "/job-service"
  REACT_APP_SCHEDULE_SERVICE_PATH: "/schedule-service"
  REACT_APP_STORAGE_SERVICE_PATH: "/storage-service"
```

```yaml
# helm-chart/react/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "react.fullname" . }}
  labels:
    {{- include "react.labels" . | nindent 4 }}
spec:
  # ... 다른 설정들 ...
  template:
    # ... metadata 설정 ...
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image:
"{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          env:
          {{- range $key, $value := .Values.env }}
            - name: {{ $key }}
              value: {{ $value | quote }}
          {{- end }}
          # ... 다른 컨테이너 설정 ...
```

3. 런타임 환경 변수 처리

React 애플리케이션이 런타임에 환경 변수에 접근할 수 있도록 ConfigMap 을 마운트하여 사용하는 방법도 있습니다:

```yaml
# helm-chart/react/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ include "react.fullname" . }}-config
  labels:
    {{- include "react.labels" . | nindent 4 }}
data:
  config.js: |
    window.ENV = {
      API_URL: "{{ .Values.env.REACT_APP_API_URL }}",
      AUTH_SERVICE_PATH:
"{{ .Values.env.REACT_APP_AUTH_SERVICE_PATH }}",
      USER_SERVICE_PATH:
"{{ .Values.env.REACT_APP_USER_SERVICE_PATH }}",
      AGENT_SERVICE_PATH:
"{{ .Values.env.REACT_APP_AGENT_SERVICE_PATH }}",
```

```
      JOB_SERVICE_PATH:
"{{ .Values.env.REACT_APP_JOB_SERVICE_PATH }}",
      SCHEDULE_SERVICE_PATH:
"{{ .Values.env.REACT_APP_SCHEDULE_SERVICE_PATH }}",
      STORAGE_SERVICE_PATH:
"{{ .Values.env.REACT_APP_STORAGE_SERVICE_PATH }}"
    };
# deployment.yaml 의 볼륨 마운트 부분
volumeMounts:
- name: config-volume
  mountPath: /usr/share/nginx/html/config.js
  subPath: config.js
volumes:
- name: config-volume
  configMap:
    name: {{ include "react.fullname" . }}-config
```

프론트엔드 환경 변수 업데이트 프로세스

프론트엔드 환경 변수를 변경해야 할 경우의 절차:

쿠버네티스 환경

1. Helm 차트의 values.yaml 파일 수정:

```
git clone https://lab.ssafy.com/s12-final/S12P31S101.git
cd S12P31S101
# values.yaml 파일 수정
git add helm-chart/react/values.yaml
git commit -m "Update frontend environment variables"
git push
```

7. ArgoCD 를 통해 변경사항 확인 및 동기화:

```
argocd app get react
argocd app sync react
```

주의사항

1. React 애플리케이션에서 환경 변수는 빌드 시점에 번들에 포함됩니다. 따라서 환경 변수가 변경되면 애플리케이션을 다시 빌드해야 합니다.

2. 민감한 정보(API 키, 비밀번호 등)는 프론트엔드 환경 변수에 포함하지 않아야 합니다. 이러한 정보는 백엔드 API 를 통해 처리하세요.
3. 빌드 시점에 포함된 환경 변수는 소스 코드와 함께 브라우저에 전송됩니다. 따라서 REACT*APP* 접두사가 있는 환경 변수는 모두 공개됩니다.
4. 런타임에 환경 설정을 변경하려면 별도의 config.js 파일을 사용하는 방식이 유용합니다.

---

6. 온프레미스 배포 (Docker 사용 x)

Python

```
# GitLab 에서 클론하시는 경우, 미리 클론을 하고 진행합니다.
# 또한 be, fe 브랜치가 나뉘어 있기에, 각 single branch 를 클론하여
진행하도록 합니다.
# 만약 master 에 합쳐져 있다면 그냥 클론하면됩니다.
# S12P31S101 는 클론한 GitLab 의 repository 이름입니다.

# backend
cd ~
mkdir be
cd be
git clone -b be --single-branch https://lab.ssafy.com/s12-
final/S12P31S101.git

# frontend
cd ~
mkdir fe
cd fe
git clone -b fe --single-branch https://lab.ssafy.com/s12-
final/S12P31S101.git

# Python 설치 확인 및 필요한 패키지 설치
sudo apt update
sudo apt install -y python3 python3-venv python3-pip

# 가상환경을 저장할 디렉토리 생성
sudo mkdir -p /opt/venvs

# 각 서비스별 가상환경 생성
for service in agent-service auth-service job-service schedule-
service storage-service; do
  sudo python3 -m venv /opt/venvs/$service
done

# 각 서비스별로 requirements 설치
for service in agent-service auth-service job-service schedule-
service; do
  echo "Installing dependencies for $service..."
  source /opt/venvs/$service/bin/activate
  pip install -r ~/be/S12P31S101/backend/$service/requirements.txt
       echo "Installing finished for $service"
  deactivate
```

```
done

# 개별 설치
# agent-service
source /opt/venvs/agent-service/bin/activate
pip install -r ~/be/S12P31S101/backend/agent-
service/requirements.txt
deactivate

# auth-service
source /opt/venvs/auth-service/bin/activate
pip install -r ~/be/S12P31S101/backend/auth-service/requirements.txt
deactivate

# job-service
source /opt/venvs/job-service/bin/activate
pip install -r ~/be/S12P31S101/backend/job-service/requirements.txt
deactivate

# schedule-service
source /opt/venvs/schedule-service/bin/activate
pip install -r ~/be/S12P31S101/backend/schedule-
service/requirements.txt
deactivate
```

Java

```
# Java 17 설치
sudo apt update
sudo apt install -y openjdk-17-jdk

# 설치 확인
java -version
```

디렉토리 구조 생성

```
# 서비스 코드와 설정을 저장할 디렉토리 생성
sudo mkdir -p /opt/services
cd /opt/services

# 각 서비스 디렉토리 생성
for service in agent-service auth-service job-service schedule-
service storage-service user-service; do
  sudo mkdir -p $service
done
```

데이터베이스

MySQL

```
# MySQL 서버 설치
sudo apt-get update
sudo apt-get install -y mysql-server

# MySQL 보안 설정
sudo mysql_secure_installation

# MySQL 서비스 시작 및 자동 시작 설정
sudo systemctl start mysql
sudo systemctl enable mysql

# 서비스별 데이터베이스 및 사용자 생성
# 임의로 설정된 계정과 비밀번호를 실제로 사용할 계정과 비밀번호로
설정해주세요.
sudo mysql -u root -p << EOF
# User Service 데이터베이스
CREATE DATABASE user_db CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
CREATE USER 'user_service'@'localhost' IDENTIFIED BY
'your_password';
GRANT ALL PRIVILEGES ON user_db.* TO 'user_service'@'localhost';

# Job Service 데이터베이스
CREATE DATABASE job_db CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
CREATE USER 'job_service'@'localhost' IDENTIFIED BY 'your_password';
GRANT ALL PRIVILEGES ON job_db.* TO 'job_service'@'localhost';

FLUSH PRIVILEGES;
EOF
```

Redis

```
# Redis 설치
sudo apt-get update
sudo apt-get install -y redis-server

# Redis 설정 수정
sudo bash -c 'cat > /etc/redis/redis.conf << EOF
# 기본 설정
port 6379
```

```
# 바인딩 주소 - 기본적으로 localhost 만 허용
bind 127.0.0.1
# 경로 설정
dir /var/lib/redis
# 데이터 지속성 설정
appendonly yes
# 메모리 제한 - 필요에 따라 조정
maxmemory 1gb
maxmemory-policy allkeys-lru
EOF'

# Redis 서비스 재시작 및 자동 시작 설정
sudo systemctl restart redis-server
sudo systemctl enable redis-server
```

Minlo

```
# MinIO 다운로드 및 설치
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
sudo mv minio /usr/local/bin/

# MinIO 서비스 계정 생성
sudo useradd -r minio-user -s /sbin/nologin

# 데이터 디렉토리 생성
sudo mkdir -p /opt/minio/data
sudo chown minio-user:minio-user /opt/minio/data

# MinIO 서비스 파일 생성
# 계정, 비밀번호 등 사용하실 정보로 설정해주시기 바랍니다.
sudo bash -c 'cat > /etc/systemd/system/minio.service << EOF
[Unit]
Description=MinIO Storage Service
Documentation=https://docs.min.io
Wants=network-online.target
After=network-online.target

[Service]
User=minio-user
Group=minio-user
Environment="MINIO_ROOT_USER=admin"
Environment="MINIO_ROOT_PASSWORD=your_minio_password"
ExecStart=/usr/local/bin/minio server /opt/minio/data --console-
address :9001
Restart=always
```

```
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOF'

# MinIO 서비스 시작 및 자동 시작 설정
sudo systemctl daemon-reload
sudo systemctl start minio
sudo systemctl enable minio
```

Milvus

! Milvus 의 경우, Docker 를 활용한 설치를 권장하며, 온프레미스 환경에서의 테스트가 어려웠던 점을 알립니다.

```
# 출처 : Milvus github README.md (https://github.com/milvus-io/milvus/blob/v2.2.9/README.md)
# Milvus 의 경우, Docker 를 활용한 설치를 권장하며, 온프레미스 환경에서의 테스트가 어려웠던 점을 알립니다.

# Clone github repository.
git clone https://github.com/milvus-io/milvus.git

sudo mv milvus /opt/

# Milvus 사용자 생성
sudo useradd -r milvus -s /sbin/nologin

# 데이터 디렉토리 생성
sudo mkdir -p /opt/milvus/data
sudo chown -R milvus:milvus /opt/milvus

# Install third-party dependencies.
cd /opt/milvus/
./scripts/install_deps.sh

# Compile Milvus.
# 컴파일 후 실행해주시기 바랍니다.
sudo make
```

AirFlow

```
# 필요한 패키지 설치
sudo apt-get update
```

```
sudo apt-get install -y build-essential libssl-dev libffi-dev
python3-dev python3-pip minio pydantic-settings openpyxl

# Airflow 전용 사용자 생성
sudo useradd -m -d /opt/airflow airflow

# Airflow 디렉토리 구조 생성
sudo mkdir -p /opt/airflow/dags /opt/airflow/logs
/opt/airflow/plugins
sudo chown -R airflow:airflow /opt/airflow

# 프로젝트의 utils 폴더 airflow/dags/utils 에 복사
sudo -u airflow cp -r /path/to/your/project/exec/utils
/opt/airflow/dags/utils

# Airflow 설치 (사용자 권한으로)
# Airflow 에 사용될 사용자 이름과 비밀번호 등을 설정해주세요.
sudo -u airflow bash -c "
export AIRFLOW_HOME=/opt/airflow
python3 -m pip install apache-airflow==2.6.3 --constraint
https://raw.githubusercontent.com/apache/airflow/constraints-
2.6.3/constraints-3.9.txt
/opt/airflow/.local/bin/airflow db init
/opt/airflow/.local/bin/airflow users create --username admin --
password your_airflow_password --firstname Admin --lastname User --
role Admin --email admin@example.com
"

# Airflow 서비스 파일 생성
sudo bash -c 'cat > /etc/systemd/system/airflow-webserver.service <<
EOF
[Unit]
Description=Airflow webserver
After=network.target

[Service]
User=airflow
Group=airflow
Type=simple
Environment="AIRFLOW_HOME=/opt/airflow"
ExecStart=/usr/local/bin/airflow webserver --port 8080
Restart=on-failure
RestartSec=5s

[Install]
WantedBy=multi-user.target
EOF'
```

```
sudo bash -c 'cat > /etc/systemd/system/airflow-scheduler.service <<
EOF
[Unit]
Description=Airflow scheduler
After=network.target

[Service]
User=airflow
Group=airflow
Type=simple
Environment="AIRFLOW_HOME=/opt/airflow"
ExecStart=/usr/local/bin/airflow scheduler
Restart=on-failure
RestartSec=5s

[Install]
WantedBy=multi-user.target
EOF'

# Airflow 서비스 시작 및 자동 시작 설정
sudo systemctl daemon-reload
sudo systemctl start airflow-webserver airflow-scheduler
sudo systemctl enable airflow-webserver airflow-scheduler
```

서비스 간 통신 설정

```
# /etc/hosts 파일 수정으로 서비스 이름 해석 설정
sudo bash -c 'cat >> /etc/hosts << EOF
127.0.0.1 agent-service
127.0.0.1 auth-service
127.0.0.1 job-service
127.0.0.1 user-service
127.0.0.1 schedule-service
127.0.0.1 milvus
127.0.0.1 airflow
127.0.0.1 redis
127.0.0.1 mysql
EOF'
```

서비스 배포

```
cd ~/be/S12P31S101/backend

# 백엔드 서비스 디렉토리로 파일 복사
for service in agent-service auth-service job-service schedule-
service storage-service; do
  sudo cp -r backend/$service/* /opt/services/$service/
```

```
done

# user-service는 Spring Boot 애플리케이션이므로 JAR 파일 빌드 후 복사
cd ~/be/S12P31S101/backend/user-service
./gradlew clean build -x test
cp build/libs/user-service*.jar /opt/services/user-service/

# 프론트엔드(React) 배포 준비
cd ~/fe/S12P31S101/frontend
# Node.js 설치 확인
if ! command -v node &> /dev/null; then
  curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
  sudo apt-get install -y nodejs
fi

# 환경변수 파일 생성
# 꼭 사용하는 환경에 맞게 설정해주시기 바랍니다.
sudo cat > .env << EOF
VITE_BASE_URL=http://k12s101.p.ssafy.io
EOF

# 의존성 설치 및 빌드
sudo npm install pnpm
pnpm install
pnpm run build

# 빌드 결과물을 서비스 디렉토리로 복사
sudo mkdir -p /opt/services/frontend
sudo cp -r dist/* /opt/services/frontend/
```

환경변수

각 서비스에 필요한 환경 변수 생성합니다.

아래 코드는 예시입니다.

실제로 사용할 설정값들을 입력해주시기 바랍니다. (DB 이름, 비밀번호, 계정 정보, URL 등)

agent-service.env

```
ALLOWED_ORIGINS=["http://localhost:5173"]
ENV=PROD
LANGSMITH_TRACING=true
```

```
LANGSMITH_ENDPOINT=https://api.smith.langchain.com
LANGSMITH_API_KEY=lsv2_pt_a0f...
LANGSMITH_PROJECT=excelerate-prod
OPENAI_API_KEY=sk-proj-ENCqyDpe9u_...
MILVUS_HOST=milvus
MILVUS_PORT=19530
MILVUS_COLLECTION=factory_catalog
FILESYSTEM_URL=http://localhost/api/storage
MINIO_ENDPOINT=minio:9000
MINIO_ACCESS_KEY=admin
MINIO_SECRET_KEY=your_minio_password
MINIO_USE_SSL=False
MINIO_BUCKET_NAME=factory-data
TOKEN_SECRET_KEY=PRODENVINJECTION
REDIS_HOST=redis
REDIS_PORT=6379
REDIS_DB=0
```

auth-service.env

```
JWT_SECRET=MjE5NDcwMjE2NTk4MjE2OTgxM3UxMDkzODJjbWE4MzAyMTg0MTIzYWJjN
zQwMjE4MzIxOTM5MDIxMjRxcGEyMTQyMTA0MTIxMjMxMjQyMTQyMTVh
```

job-service.env

```
DB_USER=root
DB_PASSWORD=root
DB_HOST=mysql
DB_PORT=3306
DB_NAME=jobs
USER_SERVICE_URL=http://user-service:8080/api/users/me/profile
```

schedule-service.env

```
USER_SERVICE_URL=http://user-service.user-
service.svc.cluster.local:8080/api/users/me/profile
JOB_SERVICE_URL=http://job-service.job-
service.svc.cluster.local:8000

AIRFLOW__CORE__DAGS_FOLDER=/opt/airflow/dags
AIRFLOW_DAGS_PATH=/opt/airflow/dags
AIRFLOW_API_URL=http://airflow-webserver.schedule-
service.svc.cluster.local:8080/api/v1
AIRFLOW_USERNAME=admin
AIRFLOW_PASSWORD=lk21ne120897ca1208
```

```
MINIO_ENDPOINT=minio.storage-service.svc.cluster.local:9000
MINIO_ACCESS_KEY=minioadmin
MINIO_SECRET_KEY=minioadmin
MINIO_USE_SSL=False
MINIO_BUCKET_NAME=factory-data

REDIS_HOST=agent-redis.agent-service.svc.cluster.local
REDIS_PORT=6379
REDIS_DB=1
REDIS_CALENDAR_CACHE_TTL=86400
```

user-service.env

```
MYSQL_HOST=user-mysql
MYSQL_PORT=3306
MYSQL_DB=userdb
MYSQL_USER=root
MYSQL_PASSWORD=root
HIBERNATE_DDL_AUTO=update
JWT_SECRET=MjE5NDcwMjE2NTk4MjE2OTgxM3UxMDkzODJjbWE4MzAyMTg0MTIzYWJjN
zQwMjE4MzIxOTM5MDIxMjRxcGEyMTQyMTA0MTIxMjMxMjQyMTQyMTVh
JWT_EXPIRATION_SECONDS=360000
NAMING_STRATEGY=SNAKE_CASE
```

서비스 설정 예시

```
# agent-service.service
sudo bash -c 'cat > /etc/systemd/system/agent-service.service << EOF
[Unit]
Description=Agent Service (FastAPI)
After=network.target

[Service]
User=serviceuser
WorkingDirectory=/opt/services/agent-service
ExecStart=/opt/venvs/agent-service/bin/uvicorn app.main:app --host
0.0.0.0 --port 8000
Environment="PYTHONPATH=/opt/services/agent-service"
EnvironmentFile=/opt/services/agent-service/agent-service.env
SuccessExitStatus=143
TimeoutStopSec=10
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF'
```

```
# auth-service.service
sudo bash -c 'cat > /etc/systemd/system/auth-service.service << EOF
[Unit]
Description=Auth Service (FastAPI)
After=network.target

[Service]
User=serviceuser
WorkingDirectory=/opt/services/auth-service
ExecStart=/opt/venvs/auth-service/bin/uvicorn main:app --host
0.0.0.0 --port 8001
Environment="PYTHONPATH=/opt/services/auth-service"
EnvironmentFile=/opt/services/auth-service/auth-service.env
SuccessExitStatus=143
TimeoutStopSec=10
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF'

# job-service.service
sudo bash -c 'cat > /etc/systemd/system/job-service.service << EOF
[Unit]
Description=Job Service (FastAPI)
After=network.target

[Service]
User=serviceuser
WorkingDirectory=/opt/services/job-service
ExecStart=/opt/venvs/job-service/bin/uvicorn app.main:app --host
0.0.0.0 --port 8002
Environment="PYTHONPATH=/opt/services/job-service"
EnvironmentFile=/opt/services/job-service/job-service.env
SuccessExitStatus=143
TimeoutStopSec=10
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF'

# schedule-service.service
sudo bash -c 'cat > /etc/systemd/system/schedule-service.service <<
EOF
```

```
[Unit]
Description=Schedule Service (FastAPI)
After=network.target

[Service]
User=serviceuser
WorkingDirectory=/opt/services/schedule-service
ExecStart=/opt/venvs/schedule-service/bin/uvicorn app.main:app --
host 0.0.0.0 --port 8003
Environment="PYTHONPATH=/opt/services/schedule-service"
EnvironmentFile=/opt/services/schedule-service/schedule-service.env
SuccessExitStatus=143
TimeoutStopSec=10
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF'

# user-service.service
sudo bash -c 'cat > /etc/systemd/system/user-service.service << EOF
[Unit]
Description=User Service (Spring Boot)
After=network.target

[Service]
User=serviceuser
WorkingDirectory=/opt/services/user-service
ExecStart=/usr/bin/java -jar /opt/services/user-service/user-
service-0.0.1-SNAPSHOT.jar
EnvironmentFile=/opt/services/user-service/user-service.env
WorkingDirectory=/opt/services/user-service
SuccessExitStatus=143
TimeoutStopSec=10
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF'
```

서비스 활성화 및 시작

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable agent-service auth-service job-service
schedule-service user-service
```

```
sudo systemctl start agent-service auth-service job-service
schedule-service user-service
```

Nginx

라우팅을 위한 Nginx 입니다.

$ 변수들은 사용하실 값을 입력해주시기 바랍니다.

```
# Nginx 설치
sudo apt-get update
sudo apt-get install -y nginx

# Nginx 설정 파일 생성
sudo bash -c 'cat > /etc/nginx/sites-available/microservices << EOF
server {
    listen 80;
    server_name localhost;

    # 프론트엔드 (React) 정적 파일 서빙
    location / {
        root /opt/services/frontend;
        try_files \$uri \$uri/ /index.html;
    }

    # 백엔드 서비스 API 라우팅
    location /api/agent-service/ {
        proxy_pass http://localhost:8000/;
        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/auth-service/ {
        proxy_pass http://localhost:8001/;
        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/job-service/ {
        proxy_pass http://localhost:8002/;
        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/schedule-service/ {
    proxy_pass http://localhost:8003/;
    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/storage-service/ {
    proxy_pass http://localhost:8004/;
    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/user-service/ {
    proxy_pass http://localhost:8080/;
    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;
}

# MinIO Console 접근
location /minio/ {
    proxy_pass http://localhost:9001/;
    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_buffering off;
    proxy_request_buffering off;
}

# Airflow UI 접근
location /airflow/ {
    proxy_pass http://localhost:8080/;
    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_buffering off;
    proxy_request_buffering off;
```

```
    }
}
EOF'

# 설정 활성화 및 Nginx 재시작
ln -s /etc/nginx/sites-available/microservices /etc/nginx/sites-
enabled/
rm -f /etc/nginx/sites-enabled/default
sudo nginx -t
sudo systemctl restart nginx
```

Port 설정

```
# UFW 를 사용하는 경우
sudo ufw allow 80/tcp    # Nginx (HTTP)
sudo ufw allow 443/tcp   # Nginx (HTTPS, 설정된 경우)
sudo ufw allow 9001/tcp  # MinIO Console
sudo ufw allow 8080/tcp  # Airflow UI

# 필요한 port 를 설정해주시기 바랍니다.
# 앞선 예시에서는 각 서비스를 8000~8003 을 사용하였습니다.
# 다른 포트를 사용하신다면 설정에 맞게 port 번호 또한 맞춰주시면 됩니다.
```

서비스 상태 확인

```
# systemd 서비스 상태 확인
sudo systemctl status \
agent-service \
auth-service \
job-service \
schedule-service \
user-service

# 서비스 로그 확인
# 예: agent-service log 확인
sudo journalctl -u agent-service -f

# 각 서비스 접근 확인
curl http://localhost/api/agent-service/health
curl http://localhost/api/auth-service/health
curl http://localhost/api/job-service/health
curl http://localhost/api/schedule-service/health
curl http://localhost/api/user-service/health

# MySQL 상태 확인
sudo systemctl status mysql
```

```
# Redis 상태 확인
sudo systemctl status redis-server
redis-cli ping  # PONG이 출력되면 정상

# Airflow 상태 확인
sudo systemctl status airflow-webserver airflow-scheduler
curl http://localhost:8080  # Airflow UI 접근 확인

# Milvus 상태 확인
sudo systemctl status milvus
```

온프레미스 환경

1. 환경 변수 업데이트 및 재빌드:

```
# 환경 변수 파일 수정
vim frontend/.env

# 재빌드 및 배포
cd frontend
npm run build
rsync -avz build/ user@deployment-server:/opt/services/frontend/
```

9. 런타임 설정 파일만 변경하는 경우:

```
# config.js 파일 수정
ssh user@deployment-server "vim /opt/services/frontend/config.js"
```

---

7. 배포 검증

🚀 운영 배포 Checklist

배포 및 운영 과정에서 다음 체크리스트를 활용하여 시스템의 정상 작동을 확인하세요:

쿠버네티스 환경

☐ Git 저장소 최신화 (values.yaml 이미지 태그 포함)
☐ Jenkins 빌드 성공 여부 확인
☐ ArgoCD 자동 동기화 상태 확인
☐ kubectl get pods --all-namespaces 이상 없음
☐ MinIO / Milvus / Airflow 포트포워딩 확인
☐ 각 서비스 기본 API 응답 확인

온프레미스 환경

☐ 각 서비스 systemd 상태 확인 (systemctl status service-name)
☐ 서비스 로그 확인 (journalctl -u service-name)
☐ MySQL 데이터베이스 연결 확인
☐ Redis 서버 응답 확인 (redis-cli ping)
☐ MinIO 서비스 확인 (curl http://localhost:9000/minio/health/live)
☐ Airflow 웹 UI 접근 확인 (http://localhost:8080)
☐ 각 서비스 기본 API 응답 확인 (curl http://localhost/api/service-name)

외부 접근을 위한 서비스 포트 구성

쿠버네티스 환경

각 관리 UI 및 서비스에 외부에서 접근하기 위한 포트 구성 방법입니다:

Airflow UI 포트 구성

```
# Airflow 웹서버를 NodePort 타입으로 변경하고 포트 40159 지정
kubectl patch svc airflow-webserver -n schedule-service -p \
'{"spec": {"type": "NodePort", "ports": [{"port": 8080,
"targetPort": 8080, "nodePort": 40159}]}}'

# 변경 확인
kubectl get svc airflow-webserver -n schedule-service
```

ArgoCD 서버 포트 구성

```
# ArgoCD 서버를 NodePort 타입으로 변경하고 포트 지정
kubectl patch svc argocd-server -n argocd -p \
'{"spec": {"type": "NodePort", "ports": [{"name": "http", "port":
80, "targetPort": 8080, "nodePort": 40157}, {"name": "https",
"port": 443, "targetPort": 8080, "nodePort": 30139}]}}'

# 변경 확인
kubectl get svc argocd-server -n argocd
```

Traefik 포트 구성

```
# Traefik 포트 확인
kubectl get svc traefik -n kube-system

# 필요시 포트 변경
kubectl patch svc traefik -n kube-system -p \
'{"spec": {"ports": [{"name": "web", "port": 80, "targetPort": 8000,
"nodePort": 30675}, {"name": "websecure", "port": 443, "targetPort":
8443, "nodePort": 31642}]}}'
```

온프레미스 환경

온프레미스 환경에서는 Nginx 가 모든 서비스에 대한 외부 접근을 처리합니다. 기본적으로 80 포트를 통해 접근할 수 있으며, 특정 서비스 포트를 직접 열고 싶은 경우 다음과 같이 방화벽을 구성합니다:

```
# UFW 를 사용하는 경우
sudo ufw allow 80/tcp    # Nginx (HTTP)
sudo ufw allow 443/tcp   # Nginx (HTTPS, 설정된 경우)
sudo ufw allow 9001/tcp  # MinIO Console
sudo ufw allow 8080/tcp  # Airflow UI

# 다른 필요한 서비스 포트 개방
```

전체 시스템 상태 확인

쿠버네티스 환경 서비스 상태 확인

```
# 모든 서비스의 포드 상태 확인
kubectl get pods --all-namespaces

# 특정 서비스의 로그 확인
kubectl logs -f deployment/agent-service -n agent-service

# 각 서비스 접근 확인 예시
curl http://172.26.10.158:30675/agent-service/
curl http://172.26.10.158:30675/auth-service/login
curl http://172.26.10.158:30675/job-service/jobs
curl http://172.26.10.158:30675/schedule-service/schedules
curl http://172.26.10.158:30675/storage-service/files
curl http://172.26.10.158:30675/user-service/users

# 서비스 포트 포워딩을 통한 직접 접근 확인
kubectl port-forward svc/agent-service -n agent-service 8000:8000
curl http://localhost:8000/
```

쿠버네티스 환경 인프라 서비스 상태 확인

Milvus 상태 확인

```
kubectl get pods -n milvus
kubectl port-forward svc/milvus -n milvus 19530:19530 &
# Milvus 클라이언트로 연결 테스트
```

MinIO 상태 확인

```
kubectl get pods -n storage-service
kubectl port-forward svc/minio -n storage-service 9000:9000 &
# 브라우저에서 http://localhost:9000 접속 (admin /
YOUR_MINIO_PASSWORD)
```

Airflow 상태 확인

```
kubectl get pods -n schedule-service
kubectl port-forward svc/airflow-webserver -n schedule-service
8080:8080 &
# 브라우저에서 http://localhost:8080 접속
```

온프레미스 환경 서비스 상태 확인

```
# systemd 서비스 상태 확인
sudo systemctl status agent-service auth-service job-service
storage-service schedule-service user-service

# 서비스 로그 확인
sudo journalctl -u agent-service -f

# 각 서비스 접근 확인
curl http://localhost/api/agent-service/health
curl http://localhost/api/auth-service/health
curl http://localhost/api/job-service/health
curl http://localhost/api/schedule-service/health
curl http://localhost/api/storage-service/health
curl http://localhost/api/user-service/health
```

온프레미스 환경 인프라 서비스 상태 확인

```
# MySQL 상태 확인
sudo systemctl status mysql

# Redis 상태 확인
sudo systemctl status redis-server
redis-cli ping  # PONG 이 출력되면 정상

# MinIO 상태 확인
sudo systemctl status minio
curl http://localhost:9001  # MinIO 콘솔 접근 확인

# Airflow 상태 확인
sudo systemctl status airflow-webserver airflow-scheduler
curl http://localhost:8080  # Airflow UI 접근 확인

# Milvus 상태 확인
sudo systemctl status milvus
```

---

8. 문제 해결

쿠버네티스 환경 문제 해결

일반적인 문제 해결 단계

- 포드 상태 확인:

```
kubectl describe pod <pod-name> -n <namespace>
```

- 로그 확인:

```
kubectl logs <pod-name> -n <namespace>
```

- 서비스 DNS 확인:

```
kubectl run -it --rm debug --image=busybox --restart=Never -- nslookup <service-name>.<namespace>.svc.cluster.local
```

- 노드 상태 확인:

```
kubectl describe node <node-name>
```

- PersistentVolumeClaim 상태 확인:

```
kubectl get pvc -n <namespace>
```

ArgoCD 관련 문제 해결

- 애플리케이션 동기화 상태 확인:

```
argocd app get <app-name>
```

- 애플리케이션 수동 동기화:

```
argocd app sync <app-name>
```

- 애플리케이션 새로고침:

```
argocd app refresh <app-name>
```

- ArgoCD 로그 확인:
- `kubectl logs -f deployment/argocd-server -n argocd`
- `kubectl logs -f deployment/argocd-repo-server -n argocd`
  `kubectl logs -f statefulset/argocd-application-controller -n argocd`

Helm 관련 문제 해결

- 릴리스 상태 확인:

```
helm status <release-name> -n <namespace>
```

- 릴리스 히스토리 확인:

```
helm history <release-name> -n <namespace>
```

- 템플릿 렌더링 결과 확인:

```
helm template <chart-path> --values <values-file> -n <namespace>
```

온프레미스 환경 문제 해결

일반적인 문제 해결 단계

- 서비스 상태 확인:

```
systemctl status <service-name>
```

- 서비스 로그 확인:

```
journalctl -u <service-name> -f
```

- 네트워크 연결 확인:

```
netstat -tulpn | grep <port>
```

- 디스크 공간 확인:

```
df -h
```

- 메모리 및 CPU 사용량 확인:

```
top
```

데이터베이스 관련 문제 해결

- MySQL 연결 확인:

```
mysql -u root -p -e "show databases;"
```

- Redis 연결 확인:

```
redis-cli ping
```

- MinIO 상태 확인:

```
curl http://localhost:9000/minio/health/live
```

웹 서버 관련 문제 해결

- Nginx 구성 파일 검증:

```
nginx -t
```

- Nginx 로그 확인:
- ```
  tail -f /var/log/nginx/error.log
  tail -f /var/log/nginx/access.log
  ```

- Nginx 재시작:

```
systemctl restart nginx
```

일반적인 오류 및 해결책

쿠버네티스 환경 오류

ImagePullBackOff 또는 ErrImagePull

증상: 포드가 ImagePullBackOff 상태로 유지됨

```
kubectl get pods -n <namespace>
# NAME                           READY   STATUS             RESTARTS
AGE
# agent-service-7d9f6b8b7-5v2zm  0/1     ImagePullBackOff   0
5m
```
```

해결책:

1. 이미지 이름과 태그가 올바른지 확인:

```
kubectl describe pod <pod-name> -n <namespace>
```

2. Docker Hub 인증 정보 확인/업데이트:

```
kubectl create secret docker-registry regcred \
  --docker-server=https://index.docker.io/v1/ \
  --docker-username=<your-name> \
  --docker-password=<your-password> \
  --docker-email=<your-email> \
  -n <namespace>
```

3. 배포에 시크릿 적용:

```
kubectl patch deployment <deployment-name> -n <namespace> \
  -p '{"spec": {"template": {"spec": {"imagePullSecrets": [{"name":
"regcred"}]}}}}'
```

CrashLoopBackOff

증상: 포드가 반복적으로 재시작됨

```
kubectl get pods -n <namespace>
# NAME                          READY   STATUS             RESTARTS
AGE
# auth-service-7d9f6b8b7-5v2zm   0/1     CrashLoopBackOff   5
5m
```

해결책:

1. 컨테이너 로그 확인:

```
kubectl logs <pod-name> -n <namespace>
```

2. 환경 변수 확인:

```
kubectl describe pod <pod-name> -n <namespace>
# Environment 섹션 확인
```

3. ConfigMap 또는 Secret 업데이트:

```
kubectl edit configmap <configmap-name> -n <namespace>
```

```
kubectl edit secret <secret-name> -n <namespace>
```

Ingress 연결 문제

증상: Ingress 경로로 접속이 안 됨

해결책:

1. Ingress 상태 확인:

```
kubectl get ingress -A
kubectl describe ingress <ingress-name> -n <namespace>
```

2. Traefik 로그 확인:

```
kubectl logs -f -l app.kubernetes.io/name=traefik -n kube-system
```

3. Ingress 클래스와 Traefik 설정 확인:

```
kubectl get ingressclass
```

온프레미스 환경 오류

서비스 시작 실패

증상: systemd 서비스가 시작되지 않음

```
systemctl status <service-name>
# Failed to start <service-name>
```

해결책:

1. 로그 확인:

```
journalctl -u <service-name> -f
```

2. 환경 변수 파일 경로 확인:

```
cat /etc/systemd/system/<service-name>.service
# EnvironmentFile=/path/to/service.env
```

3. 필요한 디렉토리 권한 확인:

```
ls -la /opt/services/<service-name>/
```

```
sudo chown -R serviceuser:serviceuser /opt/services/<service-name>/
```

API 연결 문제

증상: 서비스 간 API 호출이 실패함

해결책:

1. 서비스 상태 확인:

```
systemctl status <service-name>
```

2. /etc/hosts 파일 확인:

```
cat /etc/hosts
# 127.0.0.1 agent-service auth-service ...
```

3. Nginx 설정 확인:

```
cat /etc/nginx/sites-available/microservices
nginx -t
```

4. API 응답 직접 테스트:

```
curl http://localhost:8000/health  # 직접 서비스 포트로 테스트
curl http://localhost/api/agent-service/health  # Nginx 프록시를 통해 테스트
```

데이터베이스 연결 오류

증상: 서비스가 데이터베이스에 연결할 수 없음

해결책:

1. MySQL 상태 확인:

```
systemctl status mysql
```

2. 데이터베이스 접속 테스트:

```
mysql -u <username> -p -h <hostname> <database>
```

3. 사용자 권한 확인:

```
mysql -u root -p
mysql> SELECT User, Host, Grant_priv FROM mysql.user WHERE
User='<username>';
mysql> SHOW GRANTS FOR '<username>'@'localhost';
```

4. 환경 변수 확인:

```
cat /opt/services/<service-name>/<service-name>.env
# DB_HOST=mysql
# DB_PORT=3306
# ...
```

---

9. 부록

환경 변수 구성

현재 시스템에서는 각 서비스별로 환경 변수 파일(.env)을 사용하여 구성을 관리하고 있습니다.

쿠버네티스 환경에서의 환경 변수 관리

환경 변수 파일에서 ConfigMap 생성

```
# 환경 변수 파일에서 ConfigMap 생성
kubectl create configmap agent-service-config --from-env-file=agent-service.env -n agent-service

# 또는 특정 키-값만 추출하여 생성
grep -v "PASSWORD\|SECRET\|KEY" agent-service.env > agent-service-config.env
kubectl create configmap agent-service-config --from-env-file=agent-service-config.env -n agent-service
```

환경 변수 파일에서 Secret 생성

```
# 민감한 정보를 포함하는 환경 변수에서 Secret 생성
grep "PASSWORD\|SECRET\|KEY" agent-service.env > agent-service-secrets.env
kubectl create secret generic agent-service-secrets --from-env-file=agent-service-secrets.env -n agent-service
```

Secret 업데이트 스크립트

시스템에는 update-secret.sh 스크립트가 있어 서비스의 시크릿을 쉽게 업데이트할 수 있습니다:

```
#!/bin/bash
SERVICE=$1
if [ -z "$SERVICE" ]; then
  echo "Usage: $0 <service-name>"
  exit 1
fi
SECRET_NAME="${SERVICE}-service-secret"
ENV_FILE="${SERVICE}-service.env"
```

```
NAMESPACE="${SERVICE}-service"
kubectl delete secret "$SECRET_NAME" -n "$NAMESPACE"
kubectl create secret generic "$SECRET_NAME" \
  --from-env-file="$ENV_FILE" \
  -n "$NAMESPACE" \
  --dry-run=client -o yaml | kubectl apply -f -
```

온프레미스 환경에서의 환경 변수 관리

환경 변수 파일 관리

온프레미스 환경에서는 각 서비스의 디렉토리에 환경 변수 파일을 저장하고, systemd 서비스 파일에서 EnvironmentFile 지시문을 통해 참조합니다.

환경 변수 업데이트 스크립트 예시

```
#!/bin/bash
# update-env.sh
SERVICE=$1
if [ -z "$SERVICE" ]; then
  echo "Usage: $0 <service-name>"
  exit 1
fi

# 환경 변수 파일 업데이트
cp ${SERVICE}-service.env /opt/services/${SERVICE}-service/

# 서비스 재시작
systemctl restart ${SERVICE}-service

echo "${SERVICE}-service 환경 변수 업데이트 및 재시작 완료"
```

스케일링 가이드

쿠버네티스 환경에서의 스케일링

필요에 따라 서비스를 수평 확장할 수 있습니다:

```
# ArgoCD를 통한 스케일링 (values.yaml 수정 후 Git 저장소에 적용)

# 또는 직접 스케일링
kubectl scale deployment user-service --replicas=3 -n user-service
```

온프레미스 환경에서의 스케일링

온프레미스 환경에서는 서비스 인스턴스를 여러 서버에 배포하고 로드 밸런서를 사용하여 트래픽을 분산할 수 있습니다:

1. 여러 서버에 동일한 서비스 설정
2. Nginx 로드 밸런싱 구성 예시:

```
upstream auth-service {
    server server1:8001;
    server server2:8001;
    server server3:8001;
}

location /api/auth-service/ {
    proxy_pass http://auth-service/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

백업 및 복원 가이드

쿠버네티스 환경에서의 백업 및 복원

데이터베이스 백업:

```
# MySQL 백업 예시
kubectl exec -it $(kubectl get pods -l app=user-mysql -n user-service -o jsonpath='{.items[0].metadata.name}') -n user-service -- mysqldump -u root -p database_name > backup.sql
```

데이터베이스 복원:

```
# MySQL 복원 예시
cat backup.sql | kubectl exec -i $(kubectl get pods -l app=user-mysql -n user-service -o jsonpath='{.items[0].metadata.name}') -n user-service -- mysql -u root -p database_name
```

온프레미스 환경에서의 백업 및 복원

데이터베이스 백업:

```
# MySQL 백업
mysqldump -u root -p user_db > user_db_backup.sql
```

```
mysqldump -u root -p job_db > job_db_backup.sql

# 환경 변수 파일 백업
tar -czf env_files_backup.tar.gz /opt/services/*/
```

데이터베이스 복원:

```
# MySQL 복원
mysql -u root -p user_db < user_db_backup.sql
mysql -u root -p job_db < job_db_backup.sql

# 환경 변수 파일 복원
tar -xzf env_files_backup.tar.gz -C /
```

배포 자동화 스크립트

온프레미스 환경 배포 자동화

Docker 와 Kubernetes 없이 배포 자동화를 위한 간단한 스크립트:

```
#!/bin/bash
# deploy.sh

# 설정 변수
FASTAPI_SERVICES=("agent-service" "auth-service" "job-service"
"schedule-service" "storage-service")
SPRING_SERVICES=("user-service")
ARTIFACT_PATH="./artifacts"
TARGET_PATH="/opt/services"
TARGET_HOST="your-server-host"
VENV_PATH="/opt/venvs"

# FastAPI 서비스 배포
for service in "${FASTAPI_SERVICES[@]}"; do
  echo "Deploying FastAPI service: $service..."

  # 코드 및 환경 파일 전송
  ssh "user@$TARGET_HOST" "mkdir -p $TARGET_PATH/$service"
  scp -r "$ARTIFACT_PATH/$service/"
"user@$TARGET_HOST:$TARGET_PATH/"
  scp "$ARTIFACT_PATH/${service}.env"
"user@$TARGET_HOST:$TARGET_PATH/"

  # 가상환경 설정 (필요시)
```

```
  ssh "user@$TARGET_HOST" "if [ ! -d '$VENV_PATH/$service' ]; then
python3 -m venv $VENV_PATH/$service; fi"
  ssh "user@$TARGET_HOST" "source $VENV_PATH/$service/bin/activate
&& pip install -r $TARGET_PATH/$service/requirements.txt"

  # 서비스 재시작
  ssh "user@$TARGET_HOST" "systemctl restart $service"

  echo "$service deployment completed."
done

# Spring Boot 서비스 배포
for service in "${SPRING_SERVICES[@]}"; do
  echo "Deploying Spring Boot service: $service..."

  # JAR 파일 및 환경 파일 전송
  scp "$ARTIFACT_PATH/${service}.jar"
"user@$TARGET_HOST:$TARGET_PATH/$service/"
  scp "$ARTIFACT_PATH/${service}.env"
"user@$TARGET_HOST:$TARGET_PATH/$service/"

  # 서비스 재시작
  ssh "user@$TARGET_HOST" "systemctl restart $service"

  echo "$service deployment completed."
done

# 프론트엔드 배포
echo "Deploying React frontend..."
cd frontend
npm run build
scp -r build/* "user@$TARGET_HOST:$TARGET_PATH/frontend/"
echo "Frontend deployment completed."

echo "All services deployed successfully."
```

이 스크립트는 FastAPI 서비스와 Spring Boot 서비스를 각각 다른 방식으로 배포합니다.

모니터링 설정 가이드

쿠버네티스 환경 모니터링

쿠버네티스 환경에서는 Prometheus와 Grafana를 사용한 모니터링 설정을 권장합니다:

```
# Prometheus Operator 설치
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
helm install prometheus prometheus-community/kube-prometheus-stack -
-namespace monitoring --create-namespace

# 포트 포워딩으로 Grafana UI 접근
kubectl port-forward svc/prometheus-grafana -n monitoring 3000:80

# 기본 인증 정보: admin / prom-operator
```

온프레미스 환경 모니터링

Prometheus와 Node Exporter를 사용한 모니터링 설정:

```
# Node Exporter 설치
wget
https://github.com/prometheus/node_exporter/releases/download/v1.3.1
/node_exporter-1.3.1.linux-amd64.tar.gz
tar xvfz node_exporter-1.3.1.linux-amd64.tar.gz
sudo mv node_exporter-1.3.1.linux-amd64/node_exporter
/usr/local/bin/

# Node Exporter 서비스 설정
sudo useradd -rs /bin/false node_exporter
sudo cat > /etc/systemd/system/node_exporter.service << EOF
[Unit]
Description=Node Exporter
After=network.target

[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter

[Install]
WantedBy=multi-user.target
EOF

sudo systemctl daemon-reload
```

```
sudo systemctl start node_exporter
sudo systemctl enable node_exporter

# Prometheus 설치
wget
https://github.com/prometheus/prometheus/releases/download/v2.36.0/p
rometheus-2.36.0.linux-amd64.tar.gz
tar xvfz prometheus-2.36.0.linux-amd64.tar.gz
cd prometheus-2.36.0.linux-amd64/

# Prometheus 설정 파일 생성
cat > prometheus.yml << EOF
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node_exporter'
    static_configs:
      - targets: ['localhost:9100']

  - job_name: 'microservices'
    static_configs:
      - targets: ['localhost:8000', 'localhost:8001',
'localhost:8002', 'localhost:8003', 'localhost:8004',
'localhost:8080']
        labels:
          group: 'production'
EOF

# Prometheus 서비스 설정
sudo useradd -rs /bin/false prometheus
sudo mkdir -p /etc/prometheus /var/lib/prometheus
sudo cp prometheus promtool /usr/local/bin/
sudo cp -r consoles/ console_libraries/ /etc/prometheus/
sudo cp prometheus.yml /etc/prometheus/

sudo cat > /etc/systemd/system/prometheus.service << EOF
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
```

```
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file=/etc/prometheus/prometheus.yml \
    --storage.tsdb.path=/var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries

[Install]
WantedBy=multi-user.target
EOF

sudo chown -R prometheus:prometheus /etc/prometheus
/var/lib/prometheus
sudo systemctl daemon-reload
sudo systemctl start prometheus
sudo systemctl enable prometheus

# Grafana 설치 (선택사항)
sudo apt-get install -y software-properties-common
sudo add-apt-repository "deb https://packages.grafana.com/oss/deb
stable main"
wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add
-
sudo apt-get update
sudo apt-get install -y grafana

sudo systemctl start grafana-server
sudo systemctl enable grafana-server
```

로깅 관리 가이드

쿠버네티스 환경 로깅

쿠버네티스 환경에서는 ELK 스택 또는 Loki를 사용한 로깅을 권장합니다:

```
# Loki 스택 설치
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update
helm install loki grafana/loki-stack --namespace monitoring
```

온프레미스 환경 로깅

ELK 스택 간소화 버전 설치:

```
# Filebeat 설치 (로그 수집기)
wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-8.6.2-amd64.deb
sudo dpkg -i filebeat-8.6.2-amd64.deb

# Filebeat 구성
sudo cat > /etc/filebeat/filebeat.yml << EOF
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /var/log/nginx/*.log
    - /var/log/mysql/*.log
    - /var/log/redis/*.log
    - /var/log/syslog
  fields:
    server: "${HOSTNAME}"

# 로그를 로컬 파일로도 출력
output.file:
  path: "/var/log/filebeat"
  filename: filebeat.json
  rotate_every_kb: 10000
  number_of_files: 7
EOF

# Filebeat 시작 및 자동 시작 설정
sudo systemctl start filebeat
sudo systemctl enable filebeat

# 로그 rotate 설정 추가
sudo cat > /etc/logrotate.d/microservices << EOF
/var/log/microservices/*.log {
    daily
    missingok
    rotate 7
    compress
    delaycompress
    notifempty
    create 640 serviceuser serviceuser
    sharedscripts
    postrotate
        systemctl reload filebeat
    endscript
}
```

```
EOF
```

보안 강화 권장 사항

인증 및 접근 제어

- JWT 토큰 만료 시간 적절히 설정 (기본값: 10 시간 = 36000 초)
- 민감한 API 에 적절한 권한 검증 추가
- 모든 서비스 간 통신에 TLS 적용 (HTTPS)

네트워크 보안

- 외부 접근이 필요한 포트만 개방
- 내부 서비스 네트워크 분리
- 방화벽 규칙 적용

```
# UFW 설정 예시
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow 22/tcp    # SSH
sudo ufw allow 80/tcp    # HTTP
sudo ufw allow 443/tcp   # HTTPS
sudo ufw enable
```

암호화 및 비밀 관리

- 환경 변수 파일 보안 강화
- ```
  # 환경 변수 파일 권한 제한
  sudo chmod 600 /opt/services/*/
  ```
- 데이터베이스 패스워드 정기적 변경
- MinIO/파일 스토리지 암호화 설정

운영 체제 보안 강화

- 정기적인 보안 업데이트 적용
- ```
  sudo apt-get update
  sudo apt-get upgrade
  ```

- 불필요한 패키지 및 서비스 제거
- 강력한 SSH 설정
- # /etc/ssh/sshd_config 수정
- PermitRootLogin no
  PasswordAuthentication no

서비스 의존성 및 시작 순서

온프레미스 환경에서 서비스의 적절한 시작 순서를 보장하기 위해 systemd 유닛 파일에 의존성을 설정합니다:

```
# 기본 인프라 서비스
cat > /etc/systemd/system/infrastructure.service << EOF
[Unit]
Description=Basic Infrastructure Services
Requires=mysql.service redis.service minio.service milvus.service
After=mysql.service redis.service minio.service milvus.service

[Service]
Type=oneshot
ExecStart=/bin/true
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
EOF

# 각 서비스의 유닛 파일에 의존성 추가
for service in agent-service auth-service job-service storage-
service user-service; do
  sudo sed -i '/\[Unit\]/a
After=infrastructure.service\nRequires=infrastructure.service'
/etc/systemd/system/${service}.service
done

# schedule-service 는 Airflow 에 의존
sudo sed -i '/\[Unit\]/a After=infrastructure.service airflow-
webserver.service airflow-
scheduler.service\nRequires=infrastructure.service airflow-
webserver.service airflow-scheduler.service'
/etc/systemd/system/schedule-service.service

sudo systemctl daemon-reload
sudo systemctl enable infrastructure.service
```