포팅 매뉴얼 (SETA 프로젝트)

1. 시스템 개요

- **프로젝트**: SETA (Save the Earth Through AI)
- **목적**: AI 에너지 절감 플랫폼 - 실시간 AI 요청 필터링 시스템
- **배포 환경**: Docker 컨테이너 + 호스트 서버(EC2) 2대
- **아키텍처**: 서버1 (웹/백엔드) + 서버2 (ML/데이터 처리)

2. 기술 스택 및 버전

2.1. 서버1 (웹/백엔드 서버 - 172.26.13.106)

서비스	위치	버전	역할
PostgreSQL	Docker 컨테이너(db)	15.14	메인 데이터베이스
Redis	Docker 컨테이너(redis)	7.4.5	캐시 및 세션
Kafka	Docker 컨테이너 (bitnami/kafka:3.7)	3.7	메시징 큐
Spring Boot	Docker 컨테이너 (backend_blue/green)	3.5.5, Java 17	백엔드 서비스
React	Docker 컨테이너(frontend)	19.1.1	SPA 프론트엔드
Nginx (프론트)	Docker 컨테이너(frontend)	1.29.1	React 정적 파일 서빙
Nginx (호스트)	Ubuntu EC2	1.18.0	Blue-Green 리버스 프록시, SSL/TLS
Jaeger	Docker 컨테이너	1.57	분산 트레이싱

2.2. 서버2 (ML/데이터 처리 서버 - 172.26.8.129)

서비스	위치	버전	역할
Jenkins	Docker 컨테이너	2.516.1	CI/CD 파이프라인
ML API (FastAPI)	Docker 컨테이너	0.114.3	BERT/LLM 추론 서버
Elasticsearch	Docker 컨테이너	8.11.0	벡터 검색 및 분석

서비스	위치	버전	역할
Apache Spark	Docker 컨테이너	3.5	실시간 데이터 처리
Spark Worker	Docker 컨테이너	3.5	분산 처리 작업자
Dozzle	Docker 컨테이너	latest	컨테이너 로그 모니터링

3. 포트 매핑 테이블

3.1. 서버1 포트 구성

서비스 / 컨테이너	호스트 포트 → 컨테이너 포트
frontend (React + Nginx)	3000 → 80
backend_green / blue (Spring Boot)	8082 → 8080 / 8081 → 8080
db (PostgreSQL)	5432 → 5432
redis	6379 → 6379
kafka	9092 → 9092, 29092 → 29092, 9093 → 9093
jaeger	16686 → 16686, 4317 → 4317, 4318 → 4318
호스트 Nginx	80 → 80, 443 → 443

3.2. 서버2 포트 구성

서비스 / 컨테이너	호스트 포트 → 컨테이너 포트
jenkins	8080 → 8080
data_ml-api_1	8000 → 8000
data_elasticsearch_1	9200 → 9200
spark-master	8081 → 8080
spark-streaming-dev	8888 → 8888, 4040 → 4040
spark-streaming-app-new	4042 → 4040, 4043 → 4041
dozzle	8989 → 8080

3.3. 서버간 연동

```
서버1 (172.26.13.106) ←→ 서버2 (172.26.8.129)
│                        │
├─ PostgreSQL:5432 ←──────┼─ ML API, Spark (데이터 저장)
├─ Redis:6379 ←───────────┼─ ML API (캐싱)
```

```
├─ Kafka:9092 ──────────────┼→ Spark (메시지 스트리밍)
└─ Spring Boot:8082 ←───────┼─ 처리 결과 반환
```

---

4. 사전 준비 사항

4.1. 공통 요구사항

- **OS**: Ubuntu 22.04 LTS (양쪽 서버)
- **필수 설치**: Docker, Docker Compose
- **네트워크**: VPC 내부 통신 가능 (172.26.x.x 대역)
- **방화벽**: 필요한 포트만 개방

4.2. 서버1 전용 요구사항

- **Nginx (호스트)**: SSL/TLS 인증서 설정
- **도메인**: seta.ai.kr

4.3. 서버2 전용 요구사항

- **Jenkins**: GitLab 연동을 위한 Access Token
- **SSH Key**: 서버 접근을 위한 PEM 키

---

5. 서버1 설치 및 실행 방법

5.1. 서버 접속

```
ssh <USER>@<SERVER1_IP>
```

5.2. Git Clone

```
cd /home/<USER>
git clone https://lab.ssafy.com/s13-bigdata-dist-sub1/S13P21A403.git S13P21A403
cd S13P21A403
```

5.3. 호스트 Nginx 설정

Nginx 설치

```
sudo apt update
sudo apt install nginx
```

설정 파일 작성

```
sudo nano /etc/nginx/conf.d/nginx.conf
```

설정 내용

```
server {
    listen 443 ssl;
    server_name seta.ai.kr www.seta.ai.kr;

    ssl_certificate /etc/letsencrypt/live/seta.ai.kr/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/seta.ai.kr/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location /api/ {
        proxy_pass http://127.0.0.1:8082;  # 활성 Backend 컨테이너 포트
        proxy_http_version 1.1;
        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 / {
        proxy_pass http://127.0.0.1:3000;  # Frontend 컨테이너 포트
        proxy_http_version 1.1;
        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;
    }
```

```
}

server {
    listen 80;
    server_name seta.ai.kr www.seta.ai.kr;

    # HTTP → HTTPS 리디렉션
    return 301 https://$host$request_uri;
}
```

5.4. Docker Compose 환경 준비

Devops 디렉토리 이동

```
cd /home/ubuntu/S13P21A403/Devops
```

환경 변수 파일 생성 (.env)

```
cat > .env <<EOF
BACKEND_IMAGE=sabinm95/backend:ad75d3ed
FRONT_IMAGE=sabinm95/frontend:ad75d3ed

DB_USER=<DB_USER>
DB_PASSWORD=<DB_PASSWORD>
DB_NAME=<DB_NAME>

JWT_SECRET=<JWT_SECRET>
JWT_ACCESSTOKENEXPIRATIONMS=<JWT_ACCESSTOKENEXPIRATIONMS>
JWT_REFRESHTOKENEXPIRATIONMS=<JWT_REFRESHTOKENEXPIRATIONMS>

GMS_OPENAI_API_KEY=<GMS_OPENAI_API_KEY>
GMS_OPENAI_BASE_URL=<GMS_OPENAI_BASE_URL>
GMS_OPENAI_COMPLETIONS_PATH=<GMS_OPENAI_COMPLETIONS_PATH>
GMS_OPENAI_MODEL=<GMS_OPENAI_MODEL>
```

```
GMS_OPENAI_TIMEOUT_MS=<GMS_OPENAI_TIMEOUT_MS>
EOF
```

Docker Compose 실행

```
docker-compose -f docker-compose.frontend.yml -f docker-compose.bac
kend.yml up -d
```

---

6. 서버2 설치 및 실행 방법

6.1. 서버 접속 및 Git Clone

```
# 서버2 접속
ssh -i J13A403T.pem ubuntu@j13a403a.p.ssafy.io

# 프로젝트 클론
cd /home/ubuntu
git clone https://lab.ssafy.com/s13-bigdata-dist-sub1/S13P21A403.git S13P
21A403
cd S13P21A40
```

6.2. Jenkins 설치 및 설정

Jenkins Docker 컨테이너 실행

```
# Jenkins 데이터 디렉토리 생성
mkdir -p ~/jenkins-data
sudo chown -R 1000:1000 ~/jenkins-data

# Jenkins 컨테이너 실행
docker run -d \
  --name jenkins \
  -p 8080:8080 \
  -p 50000:50000 \
  -v ~/jenkins-data:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock:ro \
```

```
  --user root \
  jenkins/jenkins:lts

# 초기 관리자 비밀번호 확인
docker logs jenkins | grep -A 5 -B 5 "password"
```

Jenkins 초기 설정

1. 브라우저에서 `http://j13a403a.p.ssafy.io:8080` 접속
2. 초기 관리자 비밀번호 입력
3. 권장 플러그인 설치
4. 관리자 계정 생성

필수 플러그인 설치

- **Manage Jenkins** → **Manage Plugins** → **Available**
- 검색하여 설치: `GitLab Plugin`, `Docker Pipeline`, `SSH Agent Plugin`

6.3. Jenkins Credentials 설정

GitLab Deploy Token 추가

```
Manage Jenkins → Manage Credentials → (global) → Add Credentials
Kind: Username with password
Username: deploy-token-user
Password: [GitLab에서 생성한 Deploy Token]
ID: gitlab-deploy-token
```

EC2 SSH Key 추가

```
Kind: SSH Username with private key
Username: ubuntu
Private Key: [J13A403T.pem 내용]
ID: ec2-ssh-key
```

환경변수 Credentials 추가

데이터베이스 연결 (서버1)

- `postgres-host`: 172.26.13.106
- `postgres-port`: 5432
- `postgres-user`: <DB_USER>
- `postgres-password`: <DB_PASSWORD>
- `postgres-db`: <DB_NAME>

Redis 연결 (서버1)

- `redis-host`: 172.26.13.106

Kafka 연결 (서버1)

- `kafka-bootstrap-servers`: 172.26.13.106:9092
- `kafka-topic-in-raw`: raw-messages
- `kafka-topic-filter-result`: filter-result
- `kafka-topic-in-llm`: llm-queue
- `kafka-topic-out-llm-delta`: llm-delta
- `kafka-topic-out-llm-done`: llm-done

Elasticsearch

- `elasticsearch-url`: http://localhost:9200

ML 모델 설정

- `embed-index-name`: seta-embeddings
- `embedding-model-path`: /app/models/bert-model
- `embed-dims`: 768
- `filter-model-path`: /app/models/filter-model

API 설정

- `api-host`: 0.0.0.0
- `api-port`: 8000
- `api-title`: SETA ML API
- `api-version`: 1.0.0
- `environment`: production

외부 API

- `gms-api-key`: <GMS_API_KEY>
- `gms-api-url`: <GMS_API_URL>

6.4. Jenkins Pipeline Job 생성

1. **New Item** → **Pipeline** → 이름: `ml-api-deploy`
2. **Build Triggers** → **Build when a change is pushed to GitLab** 체크
3. **Pipeline** 설정:
    - Definition: `Pipeline script from SCM`
    - SCM: `Git`
    - Repository URL: `https://lab.ssafy.com/s13-bigdata-dist-sub1/S13P21A403.git`
    - Credentials: `gitlab-deploy-token`
    - Branch Specifier: `/master`
    - Script Path: `Jenkinsfile`

6.5. GitLab Webhook 설정

GitLab Repository → **Settings** → **Webhooks**

```
URL: http://j13a403a.p.ssafy.io:8080/project/ml-api-deploy
Trigger: Push events (master 브랜치)
Secret token: [Jenkins에서 생성]
SSL verification: 비활성화
```

6.6. Jenkinsfile 구성

프로젝트 루트에 `Jenkinsfile` 생성:

```
pipeline {
    agent any

    environment {
        DEPLOY_DIR = "/home/ubuntu/S13P21A403"

        POSTGRES_HOST = credentials('postgres-host')
```

```
        POSTGRES_PORT = credentials('postgres-port')
        POSTGRES_USER = credentials('postgres-user')
        POSTGRES_PASSWORD = credentials('postgres-password')
        POSTGRES_DB = credentials('postgres-db')
        REDIS_HOST = credentials('redis-host')
        ELASTICSEARCH_URL = credentials('elasticsearch-url')
        KAFKA_BOOTSTRAP_SERVERS = credentials('kafka-bootstrap-server
s')
        KAFKA_TOPIC_IN_RAW = credentials('kafka-topic-in-raw')
        KAFKA_TOPIC_FILTER_RESULT = credentials('kafka-topic-filter-resul
t')
        KAFKA_TOPIC_IN_LLM = credentials('kafka-topic-in-llm')
        KAFKA_TOPIC_OUT_LLM_DELTA = credentials('kafka-topic-out-llm-del
ta')
        KAFKA_TOPIC_OUT_LLM_DONE = credentials('kafka-topic-out-llm-do
ne')
        API_HOST = credentials('api-host')
        API_PORT = credentials('api-port')
        EMBED_INDEX_NAME = credentials('embed-index-name')
        EMBED_MODEL_PATH = credentials('embedding-model-path')
        EMBED_DIMS = credentials('embed-dims')
        FILTER_MODEL_PATH = credentials('filter-model-path')
        GMS_API_KEY = credentials('gms-api-key')
        GMS_API_URL = credentials('gms-api-url')
    }

    stages {
        stage('Deploy ML API to EC2') {
            steps {
                echo "Deploying ML API to EC2 via SSH"
                sshagent(['ec2-ssh-key']) {
                    sh '''
                        scp -o StrictHostKeyChecking=no -r ${WORKSPACE}/Data/ \
                                ${WORKSPACE}/Spark/ \
                                ubuntu@172.26.8.129:${DEPLOY_DIR}/

                        ssh -o StrictHostKeyChecking=no ubuntu@172.26.8.129 "
                            cd ${DEPLOY_DIR}/Data &&
```

```
                    echo 'Creating .env file from Jenkins credentials...' &&
                    echo \\"POSTGRES_HOST=${POSTGRES_HOST}\\" > .env
&&
                    echo \\"POSTGRES_PORT=${POSTGRES_PORT}\\" >> .env
&&
                    echo \\"POSTGRES_USER=${POSTGRES_USER}\\" >> .env
&&
                    echo \\"POSTGRES_PASSWORD=${POSTGRES_PASSWOR
D}\\" >> .env &&
                    echo \\"POSTGRES_DB=${POSTGRES_DB}\\" >> .env &&
                    echo \\"REDIS_HOST=${REDIS_HOST}\\" >> .env &&
                    echo \\"REDIS_PORT=6379\\" >> .env &&
                    echo \\"ELASTICSEARCH_URL=${ELASTICSEARCH_URL}
\\" >> .env &&
                    echo \\"KAFKA_BOOTSTRAP_SERVERS=${KAFKA_BOOTS
TRAP_SERVERS}\\" >> .env &&
                    echo \\"KAFKA_TOPIC_IN_RAW=${KAFKA_TOPIC_IN_RAW}
\\" >> .env &&
                    echo \\"KAFKA_TOPIC_FILTER_RESULT=${KAFKA_TOPIC_F
ILTER_RESULT}\\" >> .env &&
                    echo \\"KAFKA_TOPIC_IN_LLM=${KAFKA_TOPIC_IN_LLM}
\\" >> .env &&
                    echo \\"KAFKA_TOPIC_OUT_LLM_DELTA=${KAFKA_TOPIC_
OUT_LLM_DELTA}\\" >> .env &&
                    echo \\"KAFKA_TOPIC_OUT_LLM_DONE=${KAFKA_TOPIC_
OUT_LLM_DONE}\\" >> .env &&

                    echo 'Stopping existing containers...' &&
                    docker-compose down --remove-orphans || true &&
                    docker container prune -f || true &&
                    docker network prune -f || true &&

                    echo 'Starting ML API containers...' &&
                    docker-compose up -d --build &&
                    sleep 15 &&
                    echo 'ML API deployment completed'
                "
            '''
```

```
            }
        }
    }

    stage('Deploy Spark Services') {
        steps {
            echo "Deploying Spark Services to EC2"
            sshagent(['ec2-ssh-key']) {
                sh '''
                    ssh -o StrictHostKeyChecking=no ubuntu@172.26.8.129 "
                        cd ${DEPLOY_DIR}/Spark &&
                        echo 'Creating .env file for Spark services...' &&
                        echo \\"POSTGRES_HOST=${POSTGRES_HOST}\\" > .env
&&
                        echo \\"POSTGRES_PORT=${POSTGRES_PORT}\\" >> .env
&&
                        echo \\"POSTGRES_USER=${POSTGRES_USER}\\" >> .env
&&
                        echo \\"POSTGRES_PASSWORD=${POSTGRES_PASSWOR
D}\\" >> .env &&
                        echo \\"POSTGRES_DB=${POSTGRES_DB}\\" >> .env &&

                        docker-compose down --remove-orphans || true &&
                        echo 'Starting Spark services...' &&
                        docker-compose up --build -d &&
                        sleep 20 &&
                        docker-compose ps
                    "
                '''
            }
        }
    }

    stage('Verify Deployment') {
        steps {
            sshagent(['ec2-ssh-key']) {
                sh '''
                    ssh -o StrictHostKeyChecking=no ubuntu@172.26.8.129 "
```

```
                    echo '=== ML API Status ===' &&
                    cd ${DEPLOY_DIR}/Data &&
                    docker-compose ps &&

                    echo '=== Spark Services Status ===' &&
                    cd ${DEPLOY_DIR}/Spark &&
                    docker-compose ps &&

                    echo '=== Health Check ===' &&
                    curl -f http://localhost:8000/health || echo 'ML API Health
check failed'
                "
            '''
        }
    }
  }

  post {
    success {
      sshagent(['ec2-ssh-key']) {
        sh '''
          ssh -o StrictHostKeyChecking=no ubuntu@172.26.8.129 "
            echo 'Cleaning up unused Docker resources...' &&
            docker image prune -f || true &&
            docker builder prune -af || true
          "
        '''
      }
    }
    failure {
      echo "Deployment failed. Check logs for details."
    }
  }
}
```

6.7. 모니터링 도구 설치 (선택사항)

Dozzle (컨테이너 로그 모니터링)

```
docker run -d \
  --name dozzle \
  -p 8989:8080 \
  -v /var/run/docker.sock:/var/run/docker.sock:ro \
  amir20/dozzle:latest

# 방화벽 허용
sudo ufw allow 8989

# 접속: http://j13a403a.p.ssafy.io:8989
```

---

7. 배포 및 운영

7.1. 자동 배포 프로세스

```
개발자 → git push origin master
    ↓
GitLab Webhook 트리거
    ↓
Jenkins Pipeline 실행:
1) Git Checkout (master 브랜치)
2) [서버2] SCP 파일 전송 (Data/, Spark/ 폴더)
3) [서버2] 환경변수 동적 생성 (.env)
4) [서버2] Docker Compose 재배포
5) [서버2] 헬스체크 및 검증
    ↓
배포 완료 (3-5분 소요)
```

7.2. 수동 배포 방법

서버1 수동 배포

```
cd /home/ubuntu/S13P21A403/Devops
docker-compose -f docker-compose.frontend.yml -f docker-compose.bac
kend.yml down
```

```
docker-compose -f docker-compose.frontend.yml -f docker-compose.bac
kend.yml up -d --build
```

서버2 수동 배포

```
# Data 스택 재배포
cd /home/ubuntu/S13P21A403/Data
docker-compose down --remove-orphans
docker-compose up -d --build

# Spark 스택 재배포
cd /home/ubuntu/S13P21A403/Spark
docker-compose down --remove-orphans
docker-compose up -d --build
```

7.3. 주요 접속 URL

서버1

- 메인 사이트: https://seta.ai.kr
- API 엔드포인트: https://seta.ai.kr/api/

서버2

- Jenkins: http://j13a403a.p.ssafy.io:8080
- ML API: http://j13a403a.p.ssafy.io:8000
- ML API Health: http://j13a403a.p.ssafy.io:8000/health
- Elasticsearch: http://j13a403a.p.ssafy.io:9200
- Spark UI: http://j13a403a.p.ssafy.io:8888
- Dozzle 로그: http://j13a403a.p.ssafy.io:8989

---

8. 트러블슈팅

8.1. 서버1 관련 문제

백엔드 컨테이너 접속 불가

```
# 컨테이너 상태 확인
docker ps

# 로그 확인
docker logs backend_blue
docker logs backend_green

# Nginx 설정 확인
sudo nginx -t
sudo systemctl restart nginx
```

Blue-Green 전환 실패

```
# 현재 활성 포트 확인
sudo lsof -i :8082
sudo lsof -i :8081

# Nginx 설정 수정
sudo nano /etc/nginx/conf.d/nginx.conf
sudo systemctl reload nginx
```

8.2. 서버2 관련 문제

Jenkins 빌드 실패

```
# Jenkins 로그 확인
docker logs jenkins

# Webhook 연결 테스트
curl -X POST http://j13a403a.p.ssafy.io:8080/project/ml-api-deploy
```

서버간 통신 문제

```
# 네트워크 연결 테스트 (서버2에서 실행)
ping 172.26.13.106  # 서버1 연결 확인
telnet 172.26.13.106 9092  # Kafka 연결
```

```
telnet 172.26.13.106 5432  # PostgreSQL 연결
telnet 172.26.13.106 6379  # Redis 연결
```

ML API 응답 느림

```
# 컨테이너 리소스 확인
docker stats

# 메모리 사용량 확인
free -h

# Elasticsearch 상태 확인
curl http://localhost:9200/_cluster/health
```

8.3. 공통 문제

포트 충돌

```
# 포트 사용 확인
ss -tlnp | grep :8000
sudo lsof -i :8000

# 충돌하는 컨테이너 종료
docker stop $(docker ps -q)
docker system prune -f
```

디스크 공간 부족

```
# 디스크 사용량 확인
df -h

# Docker 리소스 정리
docker image prune -f
docker container prune -f
docker volume prune -f
docker network prune -f
```

---

9. 백업 및 복구

9.1. 서버1 백업

```
# PostgreSQL 백업
docker exec db pg_dump -U <DB_USER> <DB_NAME> > backup_$(date
+%Y%m%d).sql

# Redis 백업
docker exec redis redis-cli SAVE
docker cp redis:/data/dump.rdb ./redis_backup_$(date +%Y%m%d).rdb
```

9.2. 서버2 백업

```
# Jenkins 데이터 백업
sudo tar -czf jenkins-backup-$(date +%Y%m%d).tar.gz ~/jenkins-data

# Elasticsearch 백업
docker run --rm -v data_elasticsearch:/data -v $(pwd):/backup alpine \
  tar czf /backup/elasticsearch-backup-$(date +%Y%m%d).tar.gz /data
```

---

10. 보안 설정

10.1. 방화벽 설정

서버1

```
sudo ufw allow 22      # SSH
sudo ufw allow 80      # HTTP
sudo ufw allow 443     # HTTPS
sudo ufw enable
```

서버2

```
sudo ufw allow 22      # SSH
sudo ufw allow 8080    # Jenkins
sudo ufw allow 8000    # ML API
```

```
sudo ufw allow 9200    # Elasticsearch
sudo ufw allow 8888    # Spark
sudo ufw allow 8989    # Dozzle
sudo ufw enable
```

10.2. SSL/TLS 인증서 (서버1)

```
# Certbot 설치
sudo apt install certbot python3-certbot-nginx

# 인증서 발급
sudo certbot --nginx -d seta.ai.kr -d www.seta.ai.kr

# 자동 갱신 설정
sudo certbot renew --dry-run
```