포팅 매뉴얼

tag

목차

✅ EC2 설정

🔧 서비스 및 컨테이너 포트 매핑 정리

서비스명	컨테이너명	내부 포트	역할 설명	특징
Backend API	jiwon9559/storycut	8080	Spring, Django 등 백엔드 API 처리	
Nginx Proxy	nginx	80:80, 443:443	클라이언트 요청을 받아 각 서비스로 라우팅	
Jenkins	jenkins	9090, 50000	CI/CD 자동화 서버 (웹 UI는 9090, 에이전트는 50000 포트 사용)	Docker 컨테이너로 띄우지 않고 EC2 서버에 바로 설치
Prometheus	prometheus	19090	시계열 기반 모니터링 시스템, 메트릭 수집 및 시각화	
Node Exporter	node-exporter	9100	호스트 시스템의 CPU, 메모리, 디스크 등 메트릭 수집	
Grafana	grafana	3000	다양한 시각화 대시보드를 제공하는 도구	

서비스명	컨테이너명	내부 포트	역할 설명	특징
			(Prometheus 등과 연동)	
Elasticsearch	elasticsearch	9200	로그 저장 및 검색 엔진, 단일 노드 클러스터	
Logstash	logstash	5044	로그 수집 및 파싱 파이프라인 도구 (Elasticsearch에 전달)	
Kibana	kibana	5601	Elasticsearch 시각화 대시보드 툴	
Redis	redis	6379	Spring Boot 캐싱 데이터 저장	
MongoDB	mongo	27017	Spring Boot 회원 접속 로그 저장	

사용된 기술 스택 및 버전

구성 요소	기술 스택	버전
Frontend	Android Studio	
	Kotlin	
Backend	Spring Boot	Spring Boot + OpenJDK 17
Algorithm	FastAPI	FastAPI 0.95.2 / python 3.8
Proxy	Nginx	1.27.5
CI/CD	Jenkins	2.504.1 LTS
Monitoring	Prometheus	3.3.1
	Node Exporter	1.9.1
	Grafana	12.0.0
Logging	Elasticsearch	8.7.0
	Logstash	8.7.0
	Kibana	8.7.0
	Filebeat	8.7.0
Database	MySQL (AWS RDS)	8.0.41
	Redis	8.0.0

구성 요소	기술 스택	버전
	MongoDB	2.5.0

📦 EC2 환경 변수 설정 정보

- 백엔드 코드 빌드 시 사용하는 환경변수

  Jenkins UI에서 아래 파일 내용을 base64 인코딩하여 Secret Text로 입력

> 💡 **base64 인코딩 방법**
>
> application-secret.yml 파일을 저장한 위치에서 git bash 터미널 창을 열고 아래의 명령어를 실행하여 출력된 값 활용
>
> ```
> base64 -w 0 application-secret.yml
> ```

```
# application-secret.yml

spring:
  datasource:
    url:
    username:
    password:
    driver-class-name: com.mysql.cj.jdbc.Driver
  data:
    mongodb:
      uri:
    redis:
      host:
      port: 6379
      password:

  security:
    oauth2:
      client:
        registration:
          google:
```

```
            client-id:
            client-secret:
            scope:
              - email
              - profile
            redirect-uri:

jwt:
  secret:
  refresh:
  access-token-validity: 3600000  # 60분 (밀리초)
  refresh-token-validity: 2592000000 # 30일 (밀리초)
```

🛠️ GitLab CI/CD 파이프라인 순서

1. **Push / Merge**
   - **설명**: 개발자가 GitLab에 코드를 푸시하거나, Merge Request(MR)를 생성합니다. 이 단계는 CI/CD 파이프라인을 트리거하는 시작점이 됩니다.
2. **Webhook 전송**
   - **설명**: GitLab은 Push 또는 Merge 이벤트를 감지하고 설정된 Webhook을 통해 파이프라인 트리거를 외부 서비스나 다른 시스템에 전달합니다.
3. **GitLab Checkout**
   - **설명**: CI/CD 파이프라인이 시작되면 GitLab Runner가 저장소에서 최신 코드를 체크아웃하여 작업을 시작합니다.
4. **Test & Build**
   - **설명**: 프로젝트의 테스트를 실행하고, 필요하다면 빌드를 수행합니다. 예를 들어, 유닛 테스트, 통합 테스트 및 빌드 스크립트를 실행합니다.
     - Jenkins에 등록된 application-secret.yml 파일에 대한 Secret Text를 디코딩하여 적용 후 빌드 진행
5. **Docker Image Build**
   - **설명**: 테스트 및 빌드가 완료된 후, Dockerfile을 기반으로 애플리케이션의 Docker 이미지를 빌드합니다. 이 이미지에는 애플리케이션 및 필요한 종속성이 포함됩니다.
6. **Docker Image Push**

- **설명**: 빌드된 Docker 이미지를 Docker Registry(예: Docker Hub, GitLab Container Registry, AWS ECR 등)에 푸시합니다. 이를 통해 다른 시스템에서 이미지를 가져와 사용할 수 있습니다.

7. **Docker Image Pull**
   - **설명**: 프로덕션 환경 또는 다른 서버에서 최신 Docker 이미지를 풀(pull)하여, 업데이트된 애플리케이션을 가져옵니다.
8. **컨테이너 교체**
   - **설명**: 최신 Docker 이미지가 풀된 후, 기존의 컨테이너를 교체하여 새로운 버전의 애플리케이션을 실행합니다. 일반적으로 이 작업은 롤링 업데이트 방식으로 진행되며, 서비스의 중단 없이 새 이미지를 배포합니다.

🛠️ EC2 내부 설정

```
#/home/ubuntu/data/nginx/default.conf

# 로그 형식 정의
log_format json_combined escape=json
  '{'
    '"time_local":"$time_local",'
    '"remote_addr":"$remote_addr",'
    '"remote_user":"$remote_user",'
    '"request":"$request",'
    '"status": "$status",'
    '"body_bytes_sent":"$body_bytes_sent",'
    '"request_time":"$request_time",'
    '"http_referrer":"$http_referer",'
    '"http_user_agent":"$http_user_agent",'
    '"upstream_response_time":"$upstream_response_time"'
  '}';


server {
    listen 80;
    server_name k12d108.p.ssafy.io;
    server_tokens off;
```

```
    client_max_body_size 1000M;

    # 액세스 로그 설정
    access_log /var/log/nginx/access.log json_combined;
    error_log /var/log/nginx/error.log;

    if ($request_method = CONNECT) {
        return 405;
    }

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$host$request_uri;
    }

    # 민감 파일(.env, .git, .htaccess, .bashrc 등) 접근 차단
    location ~ /\.(env|git|ht|.*rc) {
        deny all;
    }
}

server {
    listen 443 ssl;
    server_name k12d108.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/k12d108.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k12d108.p.ssafy.io/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    client_max_body_size 1000M;

    if ($request_method = CONNECT) {
        return 405;
    }
```

```
#     # Kibana UI에 대한 경로 - /kibana/ 경로로 접근
    location /kibana/ {
      # /kibana/ 경로를 제거하고 프록시 전달
      # rewrite ^/kibana/(.*) /$1 break;

      proxy_pass http://kibana:5601;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_cache_bypass $http_upgrade;
      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;

      # Kibana는 대시보드 로딩에 시간이 걸릴 수 있으므로 타임아웃 설정
      proxy_read_timeout 90s;
      proxy_connect_timeout 90s;
      proxy_send_timeout 90s;
    }

    # OAuth2 콜백 경로에 대한 명시적 설정 추가
    location /api/v1/spring/login/oauth2/code/google {
        proxy_pass http://data-springboot-1:8080/api/v1/spring/login/oauth2/cod

        # 기본 헤더
        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;

        # 쿠키 관련 헤더 추가 (OAuth2 상태 관리에 중요)
        proxy_set_header Cookie $http_cookie;

        # OAuth2 리다이렉트를 위한 타임아웃 설정
        proxy_read_timeout 90s;
    }
```

```
    # OAuth2 콜백 경로에 대한 명시적 설정 추가
    location /api/v1/spring/oauth2/authorization/google {
        proxy_pass http://data-springboot-1:8080/api/v1/spring/oauth2/authoriza
        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/v1/spring/oauth2/ {
        proxy_pass http://data-springboot-1:8080/api/v1/spring/oauth2/;
        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 Cookie $http_cookie;  # 쿠키 헤더 추가 필요
    }

  location /api/v1/fastapi/ {
      proxy_pass http://hskwak.iptime.org:21201/;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-Prefix /api/v1/fastapi;
  }

  location /api/v1/spring/ {
      proxy_pass http://data-springboot-1:8080;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
  }

  # 민감 파일(.env, .git, .htaccess, .bashrc 등) 접근 차단
  location ~ /\.(env|git|ht|.*rc) {
      deny all;
  }
}
```

🛠️ 서비스 설정 (Backend, Frontend, Algorithm, Redis, Nginx)

```
#/home/ubuntu/data/docker-compose.yml

services:
  springboot:
    image: jiwon9559/storycut:latest
    ports:
      - "8080:8080"
    restart: always
    environment:
      - SPRING_PROFILES_ACTIVE=prod,secret
      - APP_BASEURL=https://k12d108.p.ssafy.io
    networks:
      - backend
    healthcheck:
      test: ["CMD", "wget", "-f", "http://localhost:8080/actuator/health"]
      interval: 30s
      timeout: 10s
      retries: 5
    volumes:
      - /home/ubuntu/log-data/springboot:/logs

  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/data/nginx:/etc/nginx/conf.d
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
      - /home/ubuntu/log-data/nginx:/var/log/nginx
    command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx -s reload; d
one & nginx -g \"daemon off;\"'"
    restart: always
    depends_on:
```

```
      - certbot
    networks:
      - backend
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 30s
      timeout: 10s
      retries: 3

  certbot:
    image: certbot/certbot
    volumes:
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h
& wait $${!}; done;'"
    restart: always
    networks:
      - backend
    healthcheck:
      test: ["CMD", "ls", "/etc/letsencrypt"]
      interval: 12h
      timeout: 10s
      retries: 1

networks:
  backend:
    name: app-network
```

📊 모니터링 서비스 (Prometheus, Node Exporter, Grafana)

```
#/home/ubuntu/monitoring/docker-compose.yml

services:
  nodeexporter:
    image: prom/node-exporter:latest
    container_name: nodeexporter
```

```
    pid: host
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro
    command:
      - '--path.procfs=/host/proc'
      - '--path.sysfs=/host/sys'
      - '--path.rootfs=/rootfs'
    ports:
      - "9100:9100"
    networks:
      - backend
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "wget", "--spider", "-q", "http://localhost:9100/metrics"]
      interval: 60s
      timeout: 10s
      retries: 5

  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - /opt/monitoring/prometheus.yml:/etc/prometheus/prometheus.yml
      - /opt/monitoring/alert.rules.yml:/etc/prometheus/alert.rules.yml
    ports:
      - "19090:9090"
    networks:
      - backend
    restart: on-failure
    healthcheck:
      test: ["CMD", "wget", "-f", "http://localhost:9090/-/healthy"]
      interval: 30s
      timeout: 10s
      retries: 3

  alertmanager:
    image: prom/alertmanager:latest
```

```
  container_name: alertmanager
  volumes:
    - /opt/monitoring/alertmanager.yml:/etc/alertmanager/config.yml
  ports:
    - "9093:9093"
  networks:
    - backend
  restart: on-failure
  healthcheck:
    test: ["CMD", "wget", "-f", "http://localhost:9093/-/healthy"]
    interval: 30s
    timeout: 10s
    retries: 3

loki:
  image: grafana/loki:2.9.3
  container_name: loki
  ports:
    - "3100:3100"
  volumes:
    - /home/ubuntu/loki-data:/loki
    - ./loki:/etc/loki
  command: -config.file=/etc/loki/loki-config.yaml
  restart: always
  networks:
    - backend

promtail:
  image: grafana/promtail:2.9.3
  container_name: promtail
  volumes:
    - /home/ubuntu/log-data/springboot:/logs       # 스프링 로그가 저장되는 곳
    - ./promtail:/etc/promtail      # promtail 설정 파일 위치
  command: -config.file=/etc/promtail/promtail-config.yml
  restart: always
  networks:
    - backend

grafana:
```

```
    image: grafana/grafana-oss:latest
    container_name: grafana
    ports:
      - "3000:3000"
    volumes:
      - /opt/monitoring/grafana:/var/lib/grafana
      - /opt/monitoring/grafana/provisioning:/etc/grafana/provisioning
    environment:
      - GF_SECURITY_ADMIN_USER=storycut
      - GF_SECURITY_ADMIN_PASSWORD=k1081234!
    networks:
      - backend
    restart: on-failure
    healthcheck:
      test: ["CMD", "wget", "-f", "http://localhost:3000/login"]
      interval: 30s
      timeout: 10s
      retries: 3

networks:
  backend:
    name: app-network
```

🔍 ELK 스택 서비스 (Elasticsearch, Logstash, Kibana, Filebeat)

```
#/home/ubuntu/elk/docker-compose.yml

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.7.0
    container_name: elasticsearch
    environment:
      - discovery.type=single-node
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms4G -Xmx4G"
      - xpack.security.enabled=false
    ulimits:
```

```
    memlock:
      soft: -1
      hard: -1
  volumes:
    - /home/ubuntu/elk-data/elasticsearch:/usr/share/elasticsearch/data
  ports:
    - 9200:9200
  networks:
    - elk
    - app-network

logstash:
  image: docker.elastic.co/logstash/logstash:8.7.0
  container_name: logstash
  volumes:
    - ./logstash/pipeline:/usr/share/logstash/pipeline
    - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml
    - /home/ubuntu/elk-data/logstash:/usr/share/logstash/data
  ports:
    - 5044:5044
    - 9600:9600
  environment:
    LS_JAVA_OPTS: "-Xmx256m -Xms256m"
  depends_on:
    - elasticsearch
  networks:
    - elk
    - app-network

kibana:
  image: docker.elastic.co/kibana/kibana:8.7.0
  container_name: kibana
  volumes:
    - /home/ubuntu/elk-data/kibana:/usr/share/kibana/data
  ports:
    - 5601:5601
  environment:
    - ELASTICSEARCH_URL=http://elasticsearch:9200
    - SERVER_BASEPATH=/kibana
```

```
      - SERVER_REWRITEBASEPATH=true
      - SERVER_PUBLICBASEURL=https://k12d108.p.ssafy.io/kibana
    depends_on:
      - elasticsearch
    networks:
      - elk
      - app-network

  filebeat:
    image: docker.elastic.co/beats/filebeat:8.7.0
    container_name: filebeat
    volumes:
      - ./filebeat/filebeat.yml:/usr/share/filebeat/filebeat.yml:ro
      - /home/ubuntu/log-data/nginx:/var/log/nginx:ro
      - /home/ubuntu/log-data/springboot:/logs:ro
    command: filebeat -e -strict.perms=false
    user: root
    depends_on:
      - elasticsearch
      - kibana
    networks:
      - elk
      - app-network

networks:
  elk:
    driver: bridge
  app-network:
    external: true
```

각각의 `docker-compose.yml` 파일이 있는 디렉터리에서 아래 명령어를 실행하면 해당 서비스들을 백그라운드에서 실행할 수 있습니다

```
docker compose up -d
```

이 명령어는 `docker-compose.yml` 파일을 참조하여 정의된 서비스들을 실행합니다. 각 파일을 처리하는 디렉터리에서 이 명령어를 실행하세요.

🦖 FastAPI 환경 설정

📁 프로젝트 폴더 구조

```
ai/
├── app/
│   ├── __init__.py
│   ├── main.py
│   ├── core/
│   │   ├── __init__.py
│   │   ├── config.py
│   │   ├── fcm.py
│   │   ├── logger.py
│   │   └── security.py
│   ├── crud/
│   │   └── __init__.py
│   ├── dependencies/
│   │   ├── __init__.py
│   │   └── s3.py
│   ├── firebase/
│   │   ├── .gitkeep
│   │   └── firebase-service-account.json
│   ├── images/
│   ├── models/
│   │   └── __init__.py
│   ├── videos/
│   └── api/
│       └── v1/
│           ├── endpoints/
│           │   ├── __init__.py
│           │   ├── mosaic.py
│           │   ├── upload.py
│           │   ├── video.py
│           │   └── video_test.py
│           ├── schemas/
│           │   ├── __init__.py
```

```
│           │   ├── mosaic_schema.py
│           │   ├── post_schema.py
│           │   ├── upload_schema.py
│           │   └── video_schema.py
│           └── services/
│               ├── __init__.py
│               ├── bgm_service.py
│               ├── mosaic_service.py
│               ├── springboot_service.py
│               ├── subtitle_service.py
│               ├── upload_service.py
│               ├── video_analysis.py
│               ├── video_edit_service.py
│               └── video_service.py
├── requirements_Anaconda.txt
├── requirements.txt
├── .gitignore
├── README.md
├── temp_clips/
└── src/
```

📦 사전 준비

✅ 1. FFmpeg 설치

- FFmpeg 공식 사이트에서 운영체제에 맞는 버전 다운로드 및 설치
- 설치 후 환경 변수(PATH)에 `ffmpeg` 실행 경로를 추가
- 설치 확인:

```
ffmpeg -version
```

✅ 2. NVIDIA 그래픽 드라이버 설치

- NVIDIA 공식 사이트에서 최신 드라이버 설치

✅ 3. CUDA 11.8 + cuDNN 설치

- CUDA 11.8 다운로드

- cuDNN for CUDA 11.8 다운로드
- 설치 확인:

```
nvcc --version
```

설치 후 환경 변수 설정

- `CUDA_HOME`, `PATH`, `LD_LIBRARY_PATH` 등을 환경변수에 추가

✅ 4. Anaconda 설치

- Anaconda 다운로드
- 설치 완료 후 아래와 같이 가상환경 구성

🐍 아나콘다 가상환경 및 라이브러리 설치

🔷 1. 가상환경 생성 (Python 3.8)

```
conda create -n mmaction2_env python=3.8 -y
```

🔷 2. 가상환경 활성화

```
conda activate mmaction2_env
```

🔷 3. PyTorch (CUDA 11.8 버전)

```
pip install torch==2.1.0 torchvision==0.16.0 torchaudio==2.1.0 --index-url <ht
tps://download.pytorch.org/whl/cu118>
```

🔷 4. MMCV 설치 (torch 2.1.0 + CUDA 11.8 호환)

```
pip install mmcv==2.1.0 -f <https://download.openmmlab.com/mmcv/dist/cu1
18/torch2.1/index.html>
```

🔷 5. 프로젝트 패키지 설치

> requirements_Anaconda.txt는 프로젝트 루트 또는 ai 폴더 안에 있어야 합니다.

```
pip install -r requirements_Anaconda.txt
```

🚀 FastAPI 서버 실행

✅ 실행 명령어

```
uvicorn app.main:app
```

> app.main:app은 main.py 파일 내부에 app = FastAPI() 선언이 있어야 작동합니다.

📁 .env 환경변수 설정

`.env` 파일은 `ai/app/` 경로 안에 위치해야 하며, 아래와 같은 형식으로 작성합니다:

```
AZURE_STORAGE_ACCOUNT_NAME=your_account_name
AZURE_STORAGE_ACCOUNT_KEY=your_account_key
AZURE_CONTAINER_NAME=your_container
GEMINI_API_KEY=your_gemini_api_key
BASE_URL=http://localhost:8080
SERVICE_ACCOUNT_PATH=app/firebase/firebase-service-account.json
```

- `firebase-service-account.json`은 Firebase 콘솔에서 발급받아 `app/firebase/` 폴더에 저장해야 합니다.

❌ 가상환경 종료

```
conda deactivate
```

✅ 체크리스트 요약

항목	완료 여부
FFmpeg 설치 및 환경변수 설정	✅
GPU 드라이버 설치	✅
CUDA 11.8 + cuDNN 설치	✅
Python 3.8 아나콘다 가상환경 구성	✅
PyTorch 및 MMCV 설치	✅
`.env` 환경변수 및 Firebase 키 설정	✅
FastAPI 실행 확인	✅