포팅메뉴얼

1. 사용 도구

- **이슈 관리**: Notion, Jira
- **형상 관리**: GitLab
- **커뮤니케이션**: MatterMost, Discord
- **디자인**: Figma
- **CI/CD**: Kubernetes + Containered, Jenkins, Docker
- **Test** : Swagger

2. 개발 도구

2.1 Frontend

- Node.js: 20.11.1
- React: 18.3.1
- TypeScript: 5.5.4
- Redux Toolkit: 2.2.7
- Axios: 1.7.7
- Styled-Component: 6.1.13
- SweetAlert2: 11.14.1

2.2 Backend

- Java: 17 (OpenJDK)
- Gradle: 7.6.4
- Springboot: 3.3.3
- JsonWebToken: 0.12.3
- Mysql: 5.7
- Redis: 7.4.0
- Swagger(OpenAPI): 2.1.0
- Locust

2.3 AI

- Python : 3.9
- Java: 1.7 (OpenJDK)
- FastAPI: 0.114.0
- APScheduler: 3.10.4
- BeautifulSoup4: 4.12.3
- konlpy: 0.6.0
- MySQLClient: 2.2.4
- Pandas: 2.2.2
- scikit-learn: 1.4.2
- Selenium: 4.24.0
- SQLAlchemy: 2.0.34
- Uvicorn: 0.30.6
- Pydantic: 2.9.1

3. K8S yaml 및 환경변수

3.1 React

react-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: react-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: react
  template:
    metadata:
      labels:
        app: react
    spec:
      containers:
        - name: react
          image: daegi0923/newscrab-front:latest  # React 애플리케이션의 Docker 이미지
          ports:
            - containerPort: 4173
          command: ["npm", "run", "preview", "--", "--host"]
          volumeMounts:
                - name: tz-seoul
                  mountPath: /etc/localtime
      volumes:
        - name: tz-seoul
          hostPath:
            path: /usr/share/zoneinfo/Asia/Seoul
```

react-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: react-service
spec:
  selector:
    app: react
  ports:
    - protocol: TCP
      port: 4173  # React 애플리케이션이 동작할 포트
      targetPort: 4173
      nodePort: 30010
  type: NodePort
```

Dockerfile

```
# Node.js 공식 이미지 사용. 경량화된 Alpine Linux 기반
FROM node:18.20.4-alpine AS build

# 작업 디렉토리 설정. 컨테이너 내 앱의 기본 경로
WORKDIR /app

# 라이브러리 설치에 필요한 파일만 복사
COPY package.json .
COPY package-lock.json .

# 라이브러리 설치
RUN npm ci

# 소스코드 복사
COPY . /app

# 소스 코드 빌드
RUN npm run build
```

3.2 Spring

- spring의 환경변수를 kubernetes의 configmap으로 관리하였습니다.

spring-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
      app: spring
  template:
    metadata:
      labels:
        app: spring
    spec:
      containers:
        - name: spring
          image: daegi0923/newscrab-back:latest  # Spring 애플리케이션의 Docker 이미지
          ports:
            - containerPort: 8080
          volumeMounts:
            - name: env-volume
              mountPath: /app/.env  # 컨테이너 내에서 마운트할 경로
              subPath: .env
          envFrom:
            - configMapRef:
                name: spring-env-config
          env:
            - name: TZ
              value: Asia/Seoul  # KST 시간대로 설정
      volumes:
        - name: env-volume
          configMap:
            name: spring-env-config
```

spring-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: spring-service
spec:
  selector:
    app: spring
  ports:
    - protocol: TCP
      port: 8080  # Spring 애플리케이션이 동작할 포트
      targetPort: 8080
      nodePort: 30020
  type: NodePort
```

spring-env-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: spring-env-config
data:
  .env: |
    # MySQL 설정
    MYSQL_HOST: mysql-service
    MYSQL_PORT: 3306
    MYSQL_DATABASE: newscrab
    MYSQL_ROOT_USER: root
    MYSQL_ROOT_PASSWORD: {MySQL Password}

    # JPA 설정
    SPRING_JPA_HIBERNATE_DDL_AUTO: none

    # Redis 설정
    REDIS_HOST: redis-service
    REDIS_PORT: 6379
    REDIS_PASSWORD: {REDIS Password}

    # 서버 설정
    SERVER_PORT: 8080

    # jwt
    JWT_SECRET : asdfhsljkdafhjjhsdauiwefbcxnlkzcblkjhsakjhdkjashdkjlabcksabkbasckb

    # 도메인 주소
    DOMAIN : https://newscrab.duckdns.org
    # ChatGPT API Key
    CHATGPT_API_KEY : {chatgpt api key 입력}
```

```
    # Fastapi 주소
    FAST_API_HOST : http://fastapi-cluster-service:8000
```

Dockerfile

```
# gradle 이미지로 빌드만
FROM gradle:7.6.4-jdk-focal AS build

WORKDIR /app

COPY build.gradle settings.gradle ./

RUN gradle dependencies --no-daemon

COPY . /app

# 빌드해서 jar 파일 생성
RUN gradle clean build --no-daemon

# JRE 이미지로 런타임 이미지 생성
FROM openjdk:17-jdk-slim

WORKDIR /app

COPY --from=build /app/build/libs/*.jar /app/newscrab.jar

EXPOSE 8080
ENTRYPOINT [ "java" ]
CMD [ "-jar", "newscrab.jar" ]
```

3.3 FastAPI

- FastAPI의 환경변수를 Kubernetes 의 configmap으로 관리하였습니다.
- FastAPI는 service 가 아닌 cluster-service로 배포하여, 외부에서 접근이 불가능합니다.
- Spring pod에서 토큰 유효성을 검사한 후 pod 내부 통신으로 접근가능하도록 하였습니다.

fastapi-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fastapi-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: fastapi
  template:
    metadata:
      labels:
        app: fastapi
    spec:
      containers:
        - name: fastapi
          image: daegi0923/newscrab-data:latest  # FastAPI 애플리케이션 Docker 이미지
          ports:
            - containerPort: 8000  # FastAPI 기본 포트
          volumeMounts:
            - name: env-volume
              mountPath: /.env  # 컨테이너 내에 .env 파일 마운트 경로
              subPath: .env
            - name: tz-seoul
              mountPath: /etc/localtime

          envFrom:
            - configMapRef:
                name: fastapi-env-config  # 환경 변수를 포함하는 ConfigMap 참조
      volumes:
        - name: env-volume
          configMap:
            name: fastapi-env-config  # .env 파일이 포함된 ConfigMap 참조
        - name: tz-seoul
          hostPath:
            path: /usr/share/zoneinfo/Asia/Seoul
```

fastapi-cluster-service.yaml

```yaml
# fastapi-cluster-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: fastapi-cluster-service
spec:
  type: ClusterIP  # ClusterIP로 설정
  selector:
    app: fastapi
  ports:
    - protocol: TCP
      port: 8000  # 내부에서 접근할 포트
      targetPort: 8000  # FastAPI 컨테이너의 포트
```

fastapi-env-config.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: fastapi-env-config
data:
  .env: |
    MYSQL_HOST=mysql-service
    MYSQL_PORT=3306
    MYSQL_ROOT_USER=root
    MYSQL_ROOT_PASSWORD={mysqlpassword}
    MYSQL_DATABASE=newscrab
```

Dockerfile

```
#
FROM python:3.9

ENV JAVA_HOME /usr/lib/jvm/java-1.7-openjdk/jre
RUN apt-get update && apt-get install -y g++ default-jdk
RUN pip install konlpy

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

RUN wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb

RUN apt install -y ./google-chrome-stable_current_amd64.deb

#
COPY ./app /code/app

RUN chmod +x /code/app/crawling/chromedriver

# FastAPI 서버 실행 (8000번 포트)
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

3.4 MySQL

mysql-statefulset.yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: "mysql-service"
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
```

```
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.7
          env:
            - name: TZ
              value: "Asia/Seoul"  # 타임존 환경 변수 설정
            - name: MYSQL_ROOT_PASSWORD
              value: "{password}"  # MySQL root 비밀번호 설정
          ports:
            - containerPort: 3306
          volumeMounts:
            - name: mysql-data
              mountPath: /var/lib/mysql
      volumes:
        - name: config-volume
          configMap:
            name: mysql-config  # 위에서 생성한 ConfigMap 참조

  volumeClaimTemplates:
    - metadata:
        name: mysql-data
      spec:
        accessModes: [ "ReadWriteOnce" ]
        resources:
          requests:
            storage: 1Gi  # 필요한 스토리지 크기 설정
```

mysql-config-map.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
data:
  my.cnf: |
    [mysqld]
    log-bin=mysql-bin  # binary log 활성화
    server-id=1
    binlog-format=row
    expire_logs_days=7
    max_binlog_size=100M
    default-time-zone='+09:00'  # 타임존을 서울(UTC+9)로 설정
```

mysql-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 1Gi  # MySQL에 필요한 스토리지 크기 설정
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data/mysql"  # 클러스터 노드의 실제 경로
```

mysql-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi  # 필요한 스토리지 크기
```

mysql-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  clusterIP: None  # Headless Service 설정
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
```

3.5 Redis

redis-statefulset.yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: redis
spec:
  serviceName: "redis-service"
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
        - name: redis
          image: redis:latest
          ports:
            - containerPort: 6379
          env:
            - name: REDIS_PASSWORD
              value: "{REDIS_Password}"  # 비밀번호 설정
          command: ["redis-server", "--requirepass", "$(REDIS_PASSWORD)"]  # 비밀번호를 설정해서 Redis 실행
          volumeMounts:
            - name: redis-data
              mountPath: /data
  volumeClaimTemplates:
    - metadata:
        name: redis-data
      spec:
        accessModes: [ "ReadWriteOnce" ]
        resources:
          requests:
            storage: 1Gi
```

redis-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-service
spec:
  selector:
    app: redis
  ports:
    - protocol: TCP
      port: 6379  # Redis 기본 포트
      targetPort: 6379
  clusterIP: None  # StatefulSet과 함께 사용할 경우, ClusterIP를 None으로 설정
```

redis-pv.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: redis-pv
spec:
```

```
  capacity:
    storage: 1Gi  # Redis에 필요한 스토리지 크기 설정
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data/redis"  # 클러스터 노드의 실제 경로
```

redis-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: redis-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi  # 필요한 스토리지 크기
```

4. 포트 설정

포트 정보

- spring : 30020
- react : 30010
- jenkins : 30000
- fastapi : 30030

nginx-config

```
server {
    listen 80;
    listen 443 ssl;
    server_name newscrab.duckdns.org;

    ssl_certificate /etc/letsencrypt/live/newscrab.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/newscrab.duckdns.org/privkey.pem;

    location /jenkins/ {
        proxy_pass http://43.202.40.103:30000;
        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-Jenkins-CLI-Host $host;  # CSRF 보호 관련 헤더 설정
        proxy_redirect off;
    }
    location /api/v1/reco/ {
        proxy_pass http://newscrab.duckdns.org:30030;  # fast-api 애플리케이션이 실행 중인 포
트
        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://newscrab.duckdns.org:30020;  # Spring 애플리케이션이 실행 중인 포트
        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;
    }
            # Swagger UI 경로
    location /swagger-ui/ {
        proxy_pass http://newscrab.duckdns.org:30020/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;
    }
```

```
    # OpenAPI 3.0 API 문서 경로
    location /v3/api-docs/ {
        proxy_pass http://newscrab.duckdns.org:30020/v3/api-docs/;
        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://newscrab.duckdns.org:30010;
            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;
        }

}
```

5. 빌드 방법

1. EC2 인스턴스 설정

- **OS 업데이트**

```
sudo apt update && sudo apt upgrade -y
```

- **필수 패키지 설치**

```
sudo apt install -y curl git
```

2. Kubernetes 설치

- 공식문서를 보고 진행했습니다.
- https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

K8S v1.31 설치

1. `apt` 패키지 인덱스 업데이트 및 Kubernetes `apt` 저장소 사용을 위한 필수 패키지 설치

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gpg
```

2. Kubernetes 패키지 저장소의 공개 서명 키 다운로드

```
# /etc/apt/keyrings 디렉토리가 없으면 생성
sudo mkdir -p -m 755 /etc/apt/keyrings

# 서명 키 다운로드 및 gpg 파일로 저장
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-
```

참고: Debian 12 및 Ubuntu 22.04 이전 릴리즈에서는 `/etc/apt/keyrings` 디렉토리가 기본적으로 존재하지 않으므로, 명령어 실행 전 해당 디렉토리를 수동으로 생성해야 합니다.

3. Kubernetes `apt` 저장소 추가

```
# 기존 설정을 덮어씁니다.
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo te
```

4. `apt` 패키지 인덱스 업데이트 후 Kubernetes 컴포넌트 설치 및 버전 고정

```
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

5. (선택 사항) kubelet 서비스 활성

```
sudo systemctl enable --now kubelet
```

포트 사용중인지 확인

```
nc 127.0.0.1 6443 -v
```

- 6443 포트가 사용중인지 확인합니다. (사용중이면 뭐 뜨는데, 사용안하고 있으면 암것도 안뜸)

컨테이너 런타임 설치

- 도커는 컨테이너 런타임 지원안됨
- containered 설치 및 시작

```
sudo apt-get update
sudo apt-get install -y containerd
sudo systemctl start containerd
sudo systemctl enable containerd
```

ip 포워딩 설정

```
sudo sysctl net.ipv4.ip_forward=1
echo "net.ipv4.ip_forward=1" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

설치 확인 후 초기화

- 싱글노드 클러스터를 구축하기 위해서 아래 명령어로 kubernetes를 초기화합니다.

```
sudo kubeadm init --control-plane-endpoint 43.202.40.103:6443 --upload-certs
```

- ip는 ec2 public ip
- kubeconfig 파일 설정

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- pod 네트워크 플러그인 설치

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

- 클러스터 상태 확인

```
kubectl get nodes
kubectl get pods --all-namespaces
```

특이사항

- K8S는 원래 마스터노드 - 워커노드를 분리하고, 마스터노드에 워크를 할당하지 못하게 설정되어 있음.
- but, EC2인스턴스가 한대밖에 없으므로 마스터노드의 taint를 해제하여 싱글노드 쿠버네티스 환경 구성

```
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
```

3. 시스템 업데이트 및 Docker 설치

```
# 1. 기존 패키지 업데이트
# 먼저 Ubuntu 패키지 관리자 `apt-get`을 사용하여 시스템의 패키지를 최신 상태로 업데이트

sudo apt-get update
sudo apt-get upgrade


# 2. Docker 설치에 필요한 패키지 설치
# Docker를 설치하기 전에, 필요한 패키지들을 먼저 설치

sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common


# 3. Docker 공식 GPG 키 추가
# Docker의 공식 GPG 키를 시스템에 추가

## 3.1 GPG 키를 다운로드 하여, `apt-key` 명령어를 통해 시스템에 추가 - 모든 APT 리포지토리에 대해 이 키가 신뢰되도록 설정
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

## 3.2 GPG 키를 특정 디렉토리에 저장하는 최신 방식 - `/etc/apt/keyrings/docker.asc` 라는 파일로 저
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

# 4. GPG 키 파일 권한 설정
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

```
# 5. Docker APT 리포지토리 추가
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# 6. 패키지 목록 업데이트
# 새로 추가된 Docker 리포지토리를 반영하기 위해 패키지 리스트를 다시 업데이트
sudo apt-get update
```

4. Nginx, Jenkins 컨테이너 배포

Docker + Nginx 설정

```
# 1. 컨테이너 실행
# HostOS 80포트를 docker container 80포트로 매핑
$ sudo docker run --name nginx -d -p 80:80 nginx

# 2. 컨테이너 쉘로 접근
$ sudo docker exec -it nginx /bin/bash

# 3. nginx 설정 파일 변경
$ sudo apt-get update
$ nano /etc/nginx/conf.d/default.conf
```

```
# 1. nginx 재시작
nginx -s reload
```

Docker + Jenkins

```
# 1. Jenkins 데이터 디렉토리 생성
$ mkdir ~/serien_project/jenkins-data

# 2. UFW 방화벽에서 포트 8080 허용
$ sudo ufw allow 8080/tcp
$ sudo ufw reload
$ sudo ufw status

# 3. Jenkins Docker 컨테이너 생성 및 실행
$ sudo docker run -d --name jenkins \
    -p 18080:8080 -p 50000:50000 \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v jenkins_home:/var/jenkins_home \
    -u root \
    jenkins/jenkins:lts
# 호스트의 8080 포트를 Jenkins 컨테이너의 8080 포트로 매핑합니다.
# 호스트의 50000 포트를 Jenkins 컨테이너의 50000 포트로 매핑합니다.
# Docker 소켓을 마운트하여 Jenkins가 Docker 명령을 실행할 수 있게 하고, jenkins_home 디렉토리를 마운트하여 Jenkins 데이터를 외부에 저장합니다.
# root 권한으로 Jenkins를 실행합니다.
```

```
docker run -d -p 30000:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home -e JENKINS_OPTS="--prefix=/jenkins" -v /var/run
```

```
# 1. jenkins 컨테이너로 접속
$ sudo docker exec -it jenkins /bin/bash

# 2. docker 설치
curl https://get.docker.com/ > dockerinstall && chmod 777 dockerinstall && ./dockerinstall
```

```
# 1. 현재 터미널 경로가 HostOS인지 확인
pwd

# 2. /var/run/docker.sock 파일에 user, group 사용자에게 읽기, 쓰기 권한 부여
sudo chmod 666 /var/run/docker.sock
```

```
# Jenkins 컨테이너 내부에서 kubectl 설치
curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-rele
chmod +x ./kubectl
mv ./kubectl /usr/local/bin/kubectl

# 설치 확인
kubectl version --client
```

Jenkins 설정

- Crediental 에 kubeconfig.yaml 파일을 등록해야합니다.
- jenkins 파이프라인에서 kubernetes의 pod을 재시작할 수 있도록 설정하기 위함

```
cat ~/.kube/config
```

- 위의 명령어로 알아낸 후, jenkins crediental 에 등록해야함
- 그래야 jenkins에서 kubernetes 명령어에 접근할 수 있다.
- Kubernetes CLI Plugin 설치 또한 필요합니다.

docker-hub, gitlab 자격증명 등록

- 이건 키 발급받아서 등록하면됨

5. Jenkins Pipelines

newscrab-pipeline-be

```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE = 'daegi0923/newscrab-back:latest'  // Docker 이미지 이름
        LOCAL_REPO_PATH = 'newscrab'  // 로컬 레포지토리 경로
        GIT_CREDENTIALS = 'gitlab-login'  // GitLab 자격증명 ID (Jenkins에 설정된 크리덴셜 ID)
        DOCKER_HUB_CREDENTIALS = 'docker-hub-login'  // Docker Hub 자격증명 ID (Jenkins에 저장된 Docker Hub 크리덴셜 ID)
        GIT_BRANCH = 'be-develop'  // 체크아웃할 Git 브랜치
        GIT_REMOTE = 'origin'  // 원격 저장소 이름
    }

    stages {
        stage('Git Clone') {
            steps {
                script {
                    git branch: "${GIT_BRANCH}", credentialsId: 'gitlab-login', url: 'https://lab.ssafy.com/s11-bigdata-reco
                }
            }
        }

        stage('Show Git Branch') {
            steps {
                script {
                    def branch = sh(script: 'git rev-parse --abbrev-ref HEAD', returnStdout: true).trim()
                    echo "Current Git Branch: ${branch}"
                }
            }
        }

        // stage('Show Directory Structure') {
        //     steps {
        //         script {
        //             sh 'find .'
        //         }
        //     }
        // }

        stage('GitLab Repository Checkout') {
            steps {
                script {
                    echo '********** GitLab Repository Checkout **********'
                    // GitLab 리포지토리에서 be-develop 브랜치 체크아웃
                    git branch: "${GIT_BRANCH}",
                        credentialsId: "${GIT_CREDENTIALS}",  // GitLab 자격 증명 ID
                        url: 'https://lab.ssafy.com/s11-bigdata-recom-sub2/S11P22E207.git'  // GitLab 리포지토리 URL
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    echo '********** Backend Build Start **********'
                    sh 'docker logout'

                    dir("${LOCAL_REPO_PATH}") {
                        // Docker 이미지 빌드
```

```
                        sh "ls -al"
                        sh "docker build -t ${DOCKER_IMAGE} ."
                    }

                    echo '********** Backend Build End **********'
                }
            }
        }

        stage('Push Docker Image to Docker Hub') {
            steps {
                script {
                    echo '********** Pushing Docker Image to Docker Hub **********'

                    // Docker Hub 로그인 (크리덴셜 사용)
                    withCredentials([usernamePassword(credentialsId: "${DOCKER_HUB_CREDENTIALS}", passwordVariable: 'DOCKER_
                        sh "echo ${DOCKER_HUB_PASSWORD} | docker login -u ${DOCKER_HUB_USERNAME} --password-stdin"
                    }

                    // Docker 이미지를 Docker Hub로 푸시
                    sh "docker push ${DOCKER_IMAGE}"
                }
            }
        }
        stage('Deploy to Kubernetes') {
            steps {
                // Kubernetes에 롤링 업데이트 적용
                withCredentials([file(credentialsId: 'kube-config', variable: 'KUBECONFIG')]) {
                sh 'kubectl get pods'
                sh 'kubectl rollout restart deployment/spring-deployment'
}

            }
        }

    }

    post {
        success {
            echo 'Pipeline completed successfully!'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

newscrab-pipeline-fe

```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE = 'daegi0923/newscrab-front:latest'  // Docker 이미지 이름
        LOCAL_REPO_PATH = 'FE-newscrab/newscrab'  // 로컬 레포지토리 경로
        GIT_CREDENTIALS = 'gitlab-login'  // GitLab 자격증명 ID (Jenkins에 설정된 크리덴셜 ID)
        DOCKER_HUB_CREDENTIALS = 'docker-hub-login'  // Docker Hub 자격증명 ID (Jenkins에 저장된 Docker Hub 크리덴셜 ID)
        GIT_BRANCH = 'fe-develop'  // 체크아웃할 Git 브랜치
        GIT_REMOTE = 'origin'  // 원격 저장소 이름
    }


    stages {
        stage('Git Clone') {
            steps {
                script {
                    git branch: "${GIT_BRANCH}", credentialsId: 'gitlab-login', url: 'https://lab.ssafy.com/s11-bigdata-reco
                }
            }
        }

        stage('Show Git Branch') {
            steps {
                script {
                    def branch = sh(script: 'git rev-parse --abbrev-ref HEAD', returnStdout: true).trim()
                    echo "Current Git Branch: ${branch}"
                }
```

```
            }
        }

        // stage('Show Directory Structure') {
        //     steps {
        //         script {
        //             sh 'find .'
        //         }
        //     }
        // }

        stage('GitLab Repository Checkout') {
            steps {
                script {
                    echo '********** GitLab Repository Checkout **********'
                    // GitLab 리포지토리에서 fe-develop 브랜치 체크아웃
                    git branch: "${GIT_BRANCH}",
                        credentialsId: "${GIT_CREDENTIALS}",  // GitLab 자격 증명 ID
                        url: 'https://lab.ssafy.com/s11-bigdata-recom-sub2/S11P22E207.git'  // GitLab 리포지토리 URL
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    echo '********** Frontend Build Start **********'
                    // sh "docker logout"
                    // sh "docker login"
                    dir("${LOCAL_REPO_PATH}") {
                        // Docker 이미지 빌드
                        sh "ls -al"
                        sh "docker build -t ${DOCKER_IMAGE} ."
                    }

                    echo '********** Frontend Build End **********'
                }
            }
        }

        stage('Push Docker Image to Docker Hub') {
            steps {
                script {
                    echo '********** Pushing Docker Image to Docker Hub **********'

                    // Docker Hub 로그인 (크리덴셜 사용)
                    withCredentials([usernamePassword(credentialsId: "${DOCKER_HUB_CREDENTIALS}", passwordVariable: 'DOCKER_
                        sh "echo ${DOCKER_HUB_PASSWORD} | docker login -u ${DOCKER_HUB_USERNAME} --password-stdin"
                    }

                    // Docker 이미지를 Docker Hub로 푸시
                    sh "docker push ${DOCKER_IMAGE}"
                }
            }
        }
        stage('Deploy to Kubernetes') {
            steps {
                // Kubernetes에 롤링 업데이트 적용
                withCredentials([file(credentialsId: 'kube-config', variable: 'KUBECONFIG')]) {
                sh 'kubectl get pods'
                sh 'kubectl rollout restart deployment/react-deployment'
              }

            }
        }
    }

    post {
        success {
            echo 'Pipeline completed successfully!'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

newscrab-pipeline-data

```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE = 'daegi0923/newscrab-data:latest'  // Docker 이미지 이름
        LOCAL_REPO_PATH = 'data'  // 로컬 레포지토리 경로
        GIT_CREDENTIALS = 'gitlab-login'  // GitLab 자격증명 ID (Jenkins에 설정된 크리덴셜 ID)
        DOCKER_HUB_CREDENTIALS = 'docker-hub-login'  // Docker Hub 자격증명 ID (Jenkins에 저장된 Docker Hub 크리덴셜 ID)
        GIT_BRANCH = 'data-develop'  // 체크아웃할 Git 브랜치
        GIT_REMOTE = 'origin'  // 원격 저장소 이름
    }


    stages {
        stage('Git Clone') {
            steps {
                script {
                    git branch: "${GIT_BRANCH}", credentialsId: 'gitlab-login', url: 'https://lab.ssafy.com/s11-bigdata-reco
                }
            }
        }

        stage('Show Git Branch') {
            steps {
                script {
                    def branch = sh(script: 'git rev-parse --abbrev-ref HEAD', returnStdout: true).trim()
                    echo "Current Git Branch: ${branch}"
                }
            }
        }

        stage('Show Directory Structure') {
            steps {
                script {
                    sh 'find .'
                }
            }
        }

        stage('GitLab Repository Checkout') {
            steps {
                script {
                    echo '********** GitLab Repository Checkout **********'
                    // GitLab 리포지토리에서 data-develop 브랜치 체크아웃
                    git branch: "${GIT_BRANCH}",
                        credentialsId: "${GIT_CREDENTIALS}",  // GitLab 자격 증명 ID
                        url: 'https://lab.ssafy.com/s11-bigdata-recom-sub2/S11P22E207.git'  // GitLab 리포지토리 URL
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    echo '********** Frontend Build Start **********'

                    dir("${LOCAL_REPO_PATH}") {
                        // Docker 이미지 빌드
                        sh "ls -al"
                        sh "docker build -t ${DOCKER_IMAGE} ."
                    }

                    echo '********** Frontend Build End **********'
                }
            }
        }

        stage('Push Docker Image to Docker Hub') {
            steps {
                script {
                    echo '********** Pushing Docker Image to Docker Hub **********'

                    // Docker Hub 로그인 (크리덴셜 사용)
                    withCredentials([usernamePassword(credentialsId: "${DOCKER_HUB_CREDENTIALS}", passwordVariable: 'DOCKER_
                        sh "echo ${DOCKER_HUB_PASSWORD} | docker login -u ${DOCKER_HUB_USERNAME} --password-stdin"
```

```
                    }

                    // Docker 이미지를 Docker Hub로 푸시
                    sh "docker push ${DOCKER_IMAGE}"
                }
            }
        }
        stage('Deploy to Kubernetes') {
            steps {
                // Kubernetes에 롤링 업데이트 적용
                withCredentials([file(credentialsId: 'kube-config', variable: 'KUBECONFIG')]) {
                sh 'kubectl get pods'
                sh 'kubectl rollout restart deployment/fastapi-deployment'
              }

            }
        }

    }

    post {
        success {
            echo 'Pipeline completed successfully!'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

6. 특이사항

- FastAPI 의 Konlpy 라이브러리가 Java 기반으로 작동하기 때문에, Dockerfile에 Java관련 추가 설정이 필요합니다.
  - 라이브러리 공식 문서 : https://konlpy.org/ko/latest/install/#docker
  - 로컬 환경에서 서버를 실행할 시, Java 환경설정이 추가로 필요
- FastAPI Dockerfile 에서 Chrome 을 설치하는 이유
  - selenium 을 사용하여 뉴스를 실시간으로 크롤링합니다.
  - selenium을 사용하려면 chrome 설치가 필요합니다.
- db password 및 chatgpt key는 바꿔서 설정해야합니다.