MSA 애플리이션 배포 - Docker Container

▼ 버전 정보

- OS: Window 10
- BE Framework: Spring Boot 3.0.10
- FE Framework: Kotlin Multiplatform
- Database: MySQL, Redis, MongoDB
- WAS: Gradle
- JVM: amazoncorretto-17
- CI/CD: Jenkins, Docker
- IDE: Intellij Ultimate, Android Studio

▼ 0. EC2 초기화 후 Docker 설치

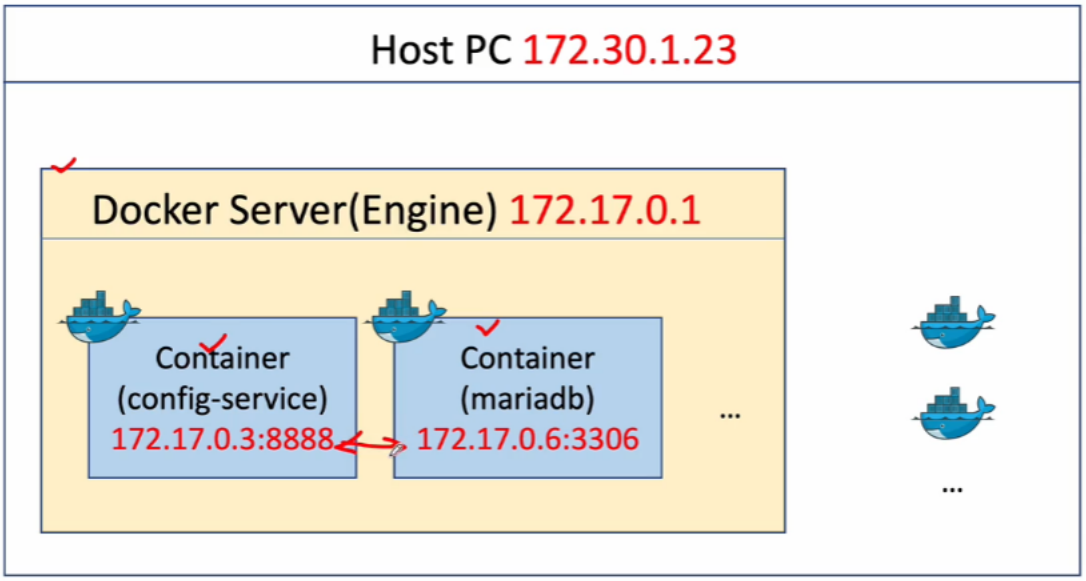
https://haengsin.tistory.com/128

```
1. 우분투 시스템 패키지 업데이트
sudo apt-get update

2. 필요한 패키지 설치
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common


3. Docker의 공식 GPG키를 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -


4. Docker의 공식 apt 저장소를 추가
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"


5. 시스템 패키지 업데이트
sudo apt-get update

6. Docker 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io


7. Docker가 설치 확인
7-1. 도커 실행상태 확인
sudo systemctl status docker

7-2. 도커 실행
sudo docker run hello-world
```

▼ 1. Network 생성

```
# 네트워크 생성
docker network create --gateway 172.18.0.1 --subnet 172.18.0.0/16 jutopia-network

# 생성한 네트워크 정보 확인
docker network inspect jutopia-network
```

- jutopia 네트워크 드라이버 생성
- 이로 인해 각각의 Microservice들은 ip address가 변경 된다 하더라도, container name으로 호출 할 수 있다



▼ 2. RabbitMQ 배포

```
docker run -d --name rabbitmq --network jutopia-network \
 -p 15672:15672 -p 5672:5672 -p 15671:15671 -p 5671:5671 -p 4369:4369 \
 -e RABBITMQ_DEFAULT_USER=guest \
 -e RABBITMQ_DEFAULT_PASS=guest rabbitmq:management
```

- 포트번호들은 rabbitMQ가 사용하는 포트번호들을 등록함
- rabbitMQ를 접속하는 기본 guest 정보를 입력

▼ 3. Config-server 배포

```
# Dockerfile
FROM amazoncorretto:17

VOLUME /tmp

COPY apiEncryptionKey.jks apiEncryptionKey.jks

COPY build/libs/config-server-1.0.jar ConfigServer.jar

ENTRYPOINT ["java", "-jar", "ConfigServer.jar"]
```

- apiEncryptionKey.jks 비대칭키

```
# 키생성
keytool -genkeypair -alias apiEncryptionKey -keyalg RSA -dname "CN=jutopia, OU=API Development, O=springprac.co.kr, L=Seould, C=KR" -key

# 키 상세보기
keytool -list -keystore apiEncryptionKey.jks -v

# bootstarp.yaml
# 로컬 key 위치
    location: file:${user.dir}/apiEncryptionKey.jks
# 컨테이너 내부
#    location: file:/apiEncryptionKey.jks
```

```
# Docker 배포

# Docker 계정으로 연동하려면/ 안하려면 서비스 네임만
docker build -t config-server:1.0 .
docker push config-server:1.0
docker pull config-server:1.0

# Docker 실행
docker run -d -p 8888:8888 --network jutopia-network \
 -e "spring.rabbitmq.host=rabbitmq" \
 -e "spring.profiles.active=default" \
 --name config-server config-server:1.0
```

- rabbitmq 컨테이너가 실행중에있어 바로 rabbitmq 서비스로 지정할 수 있다

▼ 4. Discovery-server 배포

```
# Dockerfile
FROM amazoncorretto:17

VOLUME /tmp

COPY build/libs/discovery-server-1.0.jar DiscoveryServer.jar

ENTRYPOINT ["java", "-jar", "DiscoveryServer.jar"]
```

```
# Docker 배포
docker build -t discovery-server:1.0 .
docker push discovery-server:1.0
docker pull discovery-server:1.0

# Docker 실행
docker run -d -p 8761:8761 --network jutopia-network \
 -e "spring.cloud.config.uri=http://config-server:8888" \
 --name discovery-server discovery-server:1.0
```

▼ 5. Apigateway-server 배포

```
# Dockerfile
FROM amazoncorretto:17

VOLUME /tmp

COPY build/libs/apigateway-server-1.0.jar ApigatewayServer.jar

ENTRYPOINT ["java", "-jar", "ApigatewayServer.jar"]
```

```
# Docker 배포
docker build -t apigateway-server:1.0 .
docker push apigateway-server:1.0
docker pull apigateway-server:1.0

# Docker 실행
docker run -d -p 8000:8000 --network jutopia-network \
 -e "spring.cloud.config.uri=http://config-server:8888" \
 -e "spring.rabbitmq.host=rabbitmq" \
 -e "eureka.client.serviceUrl.defaultZone=http://discovery-server:8761/eureka/" \
 --name apigateway-server \
 apigateway-server:1.0
```

※ docker 실행 시 환경변수 (-e) 작업은 해당 마이크로서비스의 yaml 파일에 적힌 내용을 적는것 이라고 보면 된다.

▼ 6. MySQL 배포

https://huisam.tistory.com/entry/mysql-replication#Container 띄워보기-1 (아래랑 다른 링크)

Mysql 8.0 replication 설정 :: TRANDENT

Spring, JSP, Javascript, JQuery, AngularJS 등 웹개발 정보 공유. Trandent.com

https://trandent.com/article/etc/detail/320833


```
# Docker 빌드
$ cd ~/S09P22108/jutopia/mysql-server
$ docker-compose up -d --build
```

(레플리케이션 용 유저 ID: repluser, PW: juto1234)

▼ master-slave 적용 전(휴지통)

```
# Docker 실행
docker run -d -p 3306:3306 --network jutopia-network --name mysqldb \
-e "SPRING_DATEASOURCE_URL=jdbc:mysql://localhost:3306/jutopia" \
-e "SPRING_DATEASOURCE_USERNAME=juto" \
-e "SPRING_DATEASOURCE_PASSWORD=juto1234" \
-e "MYSQL_ROOT_PASSWORD=juto1234" \
mysql:8.0.17

# MySQL 실행
docker exec -it -e LC_ALL=C.UTF-8 mysqldb bash

mysql -uroot -p
pw: juto1234
```

※ docker 실행 시 환경변수 (-e) 작업은 해당 마이크로서비스의 yaml 파일에 적힌 내용을 적는것 이라고 보면 된다.

▼ 7. Kafka 배포

▼ 실패 버전

```
# git clone
https://github.com/wurstmeister/kafka-docker

# docker-compose-single-broker.yml 수정
version: '2'
services:
  zookeeper-1:
    image: wurstmeister/zookeeper
    container_name: zookeeper-1
```

```
    ports:
      - "2181:2181"
    networks:
      jutopia-network:
        ipv4_address: 172.18.0.100

  zookeeper-2:
    image: wurstmeister/zookeeper
    container_name: zookeeper-2
    ports:
      - "2182:2181"
    networks:
      jutopia-network:
        ipv4_address: 172.18.0.101

  zookeeper-3:
    image: wurstmeister/zookeeper
    container_name: zookeeper-3
    ports:
      - "2183:2181"
    networks:
      jutopia-network:
        ipv4_address: 172.18.0.102

  kafka-1:
    image: wurstmeister/kafka
    container_name: kafka-1
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 172.18.0.110
      KAFKA_CREATE_TOPICS: "test:1:1"
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper-1:2181,zookeeper-2:2181,zookeeper-3:2181"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    networks:
      jutopia-network:
        ipv4_address: 172.18.0.110

  kafka-2:
    image: wurstmeister/kafka
    container_name: kafka-2
    ports:
      - "9093:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 172.18.0.111
      KAFKA_CREATE_TOPICS: "test:1:1"
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper-1:2181,zookeeper-2:2181,zookeeper-3:2181"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    networks:
      jutopia-network:
        ipv4_address: 172.18.0.111

  kafka-3:
    image: wurstmeister/kafka
    container_name: kafka-3
    ports:
      - "9094:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 172.18.0.112
      KAFKA_CREATE_TOPICS: "test:1:1"
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper-1:2181,zookeeper-2:2181,zookeeper-3:2181"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    networks:
```

```
      jutopia-network:
        ipv4_address: 172.18.0.112

networks:
  jutopia-network:
    external: true

kafka-ui:
    image: provectuslabs/kafka-ui
    container_name: kafka-ui
    ports:
      - "10000:8080"
    restart: always
    environment:
      - KAFKA_CLUSTERS_0_NAME=local
      - KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS=host.docker.internal:9092,host.docker.internal:9093,host.docker.internal:9094
      - KAFKA_CLUSTERS_0_ZOOKEEPER=zk1:2181,zk2:2182,zk1:2183
# docker-compose 실행
docker-compose -f docker-compose-single-broker.yml up -d
```

▼ 현재 사용중

[kafka] multi-node zookeeper & kafka docker-compose.yml file

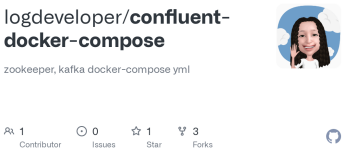
주키퍼와 카프카를 도커에 구축하기 위해서는 도커로 하나씩 실행하는 것보단 도커 컴포즈나 도커 스왐으로 실행하는 것을 추천한다. 주키퍼, 카프카 모두 분산 처리 시스템이기 때문에 여러개의 어플리케이션을 한꺼번에 띄우고 관리하는 것은 도커 컴포즈가 편하기 때문이다. 도커 컴포즈를 사용하다보면 도커 스왐이나 쿠버네티스가 더 편할것

https://log-laboratory.tistory.com/238

GitHub - logdeveloper/confluent-docker-compose: zookeeper, kafka docker-compose yml

zookeeper, kafka docker-compose yml. Contribute to logdeveloper/confluent-docker-compose development by creating an account on GitHub.

https://github.com/logdeveloper/confluent-docker-compose



- 일단 /home/broker/conf/ 해당 경로에 server1.properties, server2, server3 을 만들어둠
- **Kafka Topics 확인**
  - docker exec -it kafka1 kafka-topics --list --zookeeper zoo1:2181 → 뜨는거 확인
- **Kafka Producer 및 Consumer 테스트**: Kafka 서버에 데이터를 보내고 데이터를 가져와 보는 것이 중요합니다. 다음과 같이 Kafka Producer 및 Consumer를 실행할 수 있습니다.
  - Kafka Producer 실행 (예제 토픽 이름은 "test"로 가정):
    - docker exec -it kafka1 kafka-console-producer --broker-list kafka1:19092 --topic test
  - Kafka Consumer 실행 (예제 토픽 이름은 "test"로 가정):
    - docker exec -it kafka2 kafka-console-consumer --bootstrap-server kafka2:29092 --topic test --from-beginning
- 구버전

```
version: "3.8"
services:
  zoo1:
    image: zookeeper:3.4.9
    container_name: zoo1
    restart: always
    hostname: zoo1
    networks:
      broker-bridge:
        ipv4_address: 172.18.0.11
    ports:
      - "2181:2181"
    environment:
        ZOO_MY_ID: 1
        ZOO_PORT: 2181
```

```
      ZOO_SERVERS: server.1=zoo1:2888:3888 server.2=zoo2:2888:3888 server.3=zoo3:2888:3888
  volumes:
    - /home/broker/zookeeper-kafka-data/zoo1/data:/data
    - /home/broker/zookeeper-kafka-data/zoo1/datalog:/datalog

zoo2:
  image: zookeeper:3.4.9
  container_name: zoo2
  hostname: zoo2
  restart: always
  networks:
    broker-bridge:
      ipv4_address: 172.18.0.12
  ports:
    - "2182:2181"
  environment:
      ZOO_MY_ID: 2
      ZOO_PORT: 2182
      ZOO_SERVERS: server.1=zoo1:2888:3888 server.2=zoo2:2888:3888 server.3=zoo3:2888:3888
  volumes:
    - /home/broker/zookeeper-kafka-data/zoo2/data:/data
    - /home/broker/zookeeper-kafka-data/zoo2/datalog:/datalog

zoo3:
  image: zookeeper:3.4.9
  container_name: zoo3
  hostname: zoo3
  restart: always
  networks:
    broker-bridge:
      ipv4_address: 172.18.0.13
  ports:
    - "2183:2181"
  environment:
      ZOO_MY_ID: 3
      ZOO_PORT: 2183
      ZOO_SERVERS: server.1=zoo1:2888:3888 server.2=zoo2:2888:3888 server.3=zoo3:2888:3888
  volumes:
    - /home/broker/zookeeper-kafka-data/zoo3/data:/data
    - /home/broker/zookeeper-kafka-data/zoo3/datalog:/datalog



kafka1:
  image: confluentinc/cp-kafka:5.5.0
  container_name: kafka1
  hostname: kafka1
  restart: always
  networks:
    broker-bridge:
      ipv4_address: 172.18.0.21
  ports:
    - "9092:9091"
  environment:
    KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka1:19092,LISTENER_DOCKER_EXTERNAL://${DOCKER_HOST_IP:-127.0.0.1}:9
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
    KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181,zoo2:2182,zoo3:2183"
    KAFKA_BROKER_ID: 1
    KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
    # replication factor
    KAFKA_DEFAULT_REPLICATION_FACTOR : 2
    # partition
    KAFKA_NUM_PARTITIONS: 3
  volumes:
    - /home/broker/zookeeper-kafka-data/kafka1/data:/var/lib/kafka/data
    # you have to create server.properties file before starting docker compose.
    - /home/broker/conf/server1.properties:/etc/kafka/server.properties
  depends_on:
    - zoo1
    - zoo2
    - zoo3

kafka2:
  image: confluentinc/cp-kafka:5.5.0
  container_name: kafka2
  hostname: kafka2
  restart: always
```

```
    networks:
      broker-bridge:
        ipv4_address: 172.18.0.32
    ports:
      - "9093:9091"
    environment:
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka2:19093,LISTENER_DOCKER_EXTERNAL://${DOCKER_HOST_IP:-127.0.0.1}:9
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181,zoo2:2182,zoo3:2183"
      KAFKA_BROKER_ID: 2
      KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
      # replication factor
      KAFKA_DEFAULT_REPLICATION_FACTOR : 2
      # partition
      KAFKA_NUM_PARTITIONS: 3
    volumes:
      - /home/broker/zookeeper-kafka-data/kafka2/data:/var/lib/kafka/data
      # you have to create server.properties file before starting docker compose.
      - /home/broker/conf/server2.properties:/etc/kafka/server.properties
    depends_on:
      - zoo1
      - zoo2
      - zoo3

  kafka3:
    image: confluentinc/cp-kafka:5.5.0
    container_name: kafka3
    hostname: kafka3
    restart: always
    networks:
      broker-bridge:
        ipv4_address: 172.18.0.33
    ports:
      - "9094:9091"
    environment:
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka3:19094,LISTENER_DOCKER_EXTERNAL://${DOCKER_HOST_IP:-127.0.0.1}:9
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181,zoo2:2182,zoo3:2183"
      KAFKA_BROKER_ID: 3
      KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
      # replication factor
      KAFKA_DEFAULT_REPLICATION_FACTOR : 2
      # partition
      KAFKA_NUM_PARTITIONS: 3
    volumes:
      - /home/broker/zookeeper-kafka-data/kafka3/data:/var/lib/kafka/data
      # you have to create server.properties file before starting docker compose.
      - /home/broker/conf/server3.properties:/etc/kafka/server.properties
    depends_on:
      - zoo1
      - zoo2
      - zoo3
  kafka-ui:
    image: provectuslabs/kafka-ui
    container_name: kafka-ui
    ports:
      - "9009:9009"
    restart: always
    networks:
      - broker-bridge
    environment:
      - KAFKA_CLUSTERS_0_NAME=local
      - KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS=kafka1:9092,kafka2:9093,kafka3:9094
      - KAFKA_CLUSTERS_0_ZOOKEEPER=zoo1:2181,zoo2:2182,zoo3:2183

networks:
  broker-bridge:
    external:
      name: jutopia-network



#_---------------------------0928 백업
version: "3"
services:
  zoo1:
```

```
    image: confluentinc/cp-zookeeper:6.2.0
    container_name: zoo1
    restart: always
    hostname: zoo1
    networks:
      broker-bridge:
        ipv4_address: 172.18.0.51
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  zoo2:
    image: confluentinc/cp-zookeeper:6.2.0
    container_name: zoo2
    hostname: zoo2
    restart: always
    networks:
      broker-bridge:
        ipv4_address: 172.18.0.52
    ports:
      - "2182:2181"
    environment:
      ZOOKEEPER_SERVER_ID: 2
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  zoo3:
    image: confluentinc/cp-zookeeper:6.2.0
    container_name: zoo3
    hostname: zoo3
    restart: always
    networks:
      broker-bridge:
        ipv4_address: 172.18.0.53
    ports:
      - "2183:2181"
    environment:
      ZOOKEEPER_SERVER_ID: 3
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka1:
    image: confluentinc/cp-kafka:6.2.0
    container_name: kafka1
    hostname: kafka1
    restart: always
    networks:
      broker-bridge:
        ipv4_address: 172.18.0.61
    ports:
      - "9092:9092"
      - "19092:19092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka1:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    depends_on:
      - zoo1
      - zoo2
      - zoo3

  kafka2:
    image: confluentinc/cp-kafka:6.2.0
    container_name: kafka2
    hostname: kafka2
    restart: always
    networks:
      broker-bridge:
        ipv4_address: 172.18.0.62
    ports:
      - "9093:9092"
      - "19093:19093"
    environment:
```

```
      KAFKA_BROKER_ID: 2
      KAFKA_ZOOKEEPER_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka2:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    depends_on:
      - zoo1
      - zoo2
      - zoo3

  kafka3:
    image: confluentinc/cp-kafka:6.2.0
    container_name: kafka3
    hostname: kafka3
    restart: always
    networks:
      broker-bridge:
        ipv4_address: 172.18.0.63
    ports:
      - "9094:9092"
      - "19094:19094"
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_ZOOKEEPER_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka3:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    depends_on:
      - zoo1
      - zoo2
      - zoo3

  kafka-ui:
    image: provectuslabs/kafka-ui
    container_name: kafka-ui
    ports:
      - "10000:8099"
    restart: always
    networks:
      - broker-bridge
    environment:
      - KAFKA_CLUSTERS_0_NAME=jutopia
      - KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS=PLAINTEXT://kafka1:9092,PLAINTEXT://kafka2:9093,PLAINTEXT://kafka3:9094
      - KAFKA_CLUSTERS_0_ZOOKEEPER=zoo1:2181,zoo2:2182,zoo3:2183
    depends_on:
      - zoo1
      - zoo2
      - zoo3
      - kafka1
      - kafka2
      - kafka3

networks:
  broker-bridge:
    external:
      name: jutopia-network
```

```
# 현재 서버 버전
version: '3'
services:
  zookeeper-1:
    container_name: zookeeper1
    hostname: zookeeper1
    networks:
      - broker-bridge
    image: confluentinc/cp-zookeeper:6.2.0
    environment:
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_CLIENT_PORT: 12181
      ZOOKEEPER_DATA_DIR: ./zookeeper/data
#22888:23888 = 주키퍼 앙상블간에 통신 포트:리더 일렉션시 사용 포트
      ZOOKEEPER_SERVERS: zookeeper1:22888:23888;zookeeper2:32888:33888;zookeeper3:42888:43888
#포트포워딩 호스트 12181로 들어오는 요청 -> 컨테이너12181
    ports:
      - 12181:12181
```

```
    - 22888:22888
    - 23888:23888
  volumes:
    - ./zookeeper/data/1:/zookeeper/data

zookeeper-2:
  container_name: zookeeper2
  hostname: zookeeper2
  networks:
    - broker-bridge
  image: confluentinc/cp-zookeeper:6.2.0
  environment:
    ZOOKEEPER_SERVER_ID: 2
    ZOOKEEPER_CLIENT_PORT: 22181
    ZOOKEEPER_DATA_DIR: ./zookeeper/data
    ZOOKEEPER_SERVERS: zookeeper1:22888:23888;zookeeper2:32888:33888;zookeeper3:42888:43888
  ports:
    - 22181:22181
    - 32888:32888
    - 33888:33888
  volumes:
    - ./zookeeper/data/2:/zookeeper/data

zookeeper-3:
  container_name: zookeeper3
  hostname: zookeeper3
  networks:
    - broker-bridge
  image: confluentinc/cp-zookeeper:6.2.0
  environment:
    ZOOKEEPER_SERVER_ID: 3
    ZOOKEEPER_CLIENT_PORT: 32181
    ZOOKEEPER_DATA_DIR: ./zookeeper/data
    ZOOKEEPER_SERVERS: zookeeper1:22888:23888;zookeeper2:32888:33888;zookeeper3:42888:43888
  ports:
    - 32181:32181
    - 42888:42888
    - 43888:43888
  volumes:
    - ./zookeeper/data/3:/zookeeper/data

kafka-1:
  image: confluentinc/cp-kafka:6.2.0
  container_name: kafka1
  hostname: kafka1
  networks:
    - broker-bridge
  depends_on:
    - zookeeper-1
    - zookeeper-2
    - zookeeper-3
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper1:12181,zookeeper2:22181,zookeeper3:32181
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka1:19092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME : PLAINTEXT
    KAFKA_LOG_DIRS: ./kafka
  ports:
    - 19092:19092
  volumes:
    - ./kafka/logs/1:/kafka

kafka-2:
  image: confluentinc/cp-kafka:6.2.0
  container_name: kafka2
  hostname: kafka2
  networks:
    - broker-bridge
  depends_on:
    - zookeeper-1
    - zookeeper-2
    - zookeeper-3
  environment:
    KAFKA_BROKER_ID: 2
    KAFKA_ZOOKEEPER_CONNECT: zookeeper1:12181,zookeeper2:22181,zookeeper3:32181
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka2:29092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
```

```
      KAFKA_INTER_BROKER_LISTENER_NAME : PLAINTEXT
      KAFKA_LOG_DIRS: ./kafka
    ports:
      - 29092:29092
    volumes:
      - ./kafka/logs/2:/kafka

  kafka-3:
    image: confluentinc/cp-kafka:6.2.0
    hostname: kafka3
    container_name: kafka3
    networks:
      - broker-bridge
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_ZOOKEEPER_CONNECT: zookeeper1:12181,zookeeper2:22181,zookeeper3:32181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka3:39092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME : PLAINTEXT
      KAFKA_LOG_DIRS: ./kafka
    ports:
      - 39092:39092
    volumes:
      - ./kafka/logs/3:/kafka


  kafka-ui:
    image: provectuslabs/kafka-ui
    container_name: kafka-ui
    ports:
      - 10000:8099
    restart: always
    networks:
      - broker-bridge
    depends_on:
      - kafka-1
      - kafka-2
      - kafka-3
    environment:
      - KAFKA_CLUSTERS_0_NAME=jutopia
      - KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS=kafka1:19092,kafka2:29092,kafka3:39092
      - KAFKA_CLUSTERS_0_ZOOKEEPERS=zookeeper1:12181, zookeeper1:22181, zookeeper1:32181

networks:
  broker-bridge:
    external:
      name: jutopia-network
```

▼ 8. Mongo-DB 배포

https://wooiljeong.github.io/server/docker-mongo/

(Dockerfile은 추후 작성)

```
# Docker 배포

$ docker pull mongo

$ sudo docker run --name mongo -v ~/data:/data/db -d -p 27017:27017 --network jutopia-network mongo

$ docker exec –it mongodb mongosh
(mongodb 대신 containerID 복붙)
```

▼ 9. News-server 배포

```
# Dockerfile
FROM python:3.11

COPY ./src/main.py ./main.py

COPY ./requirements.txt ./requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 9091

CMD ["uvicorn", "main:app", "--host=0.0.0.0", "--port=9091"]
```

```
# Docker 빌드
$ cd ~/S09P22C108/jutopia/news-server/
$ sudo docker build -t news-server:1.0 .

# Docker 배포
$ docker run -d --name news-server -p 9001:9001 --network jutopia-network \
news-server:1.0
위 명령어가 안되면 ps에서 돌아가고 있는것
```

▼ 10. Class-server 배포

```
# Dockerfile
FROM amazoncorretto:17

VOLUME /tmp

COPY build/libs/class-server-1.0.jar ClassServer.jar

ENTRYPOINT ["java", "-jar", "ApigatewayServer.jar"]
```

```
# Docker 빌드
docker build -t class-server:1.0 .

# Docker 배포
docker run -d --network jutopia-network \
 --name class-server \
 -e "spring.cloud.config.uri=http://config-server:8888" \
 -e "spring.rabbitmq.host=rabbitmq" \
 -e "spring.zipkin.base-url=http://zipkin:9411" \
 -e "eureka.client.serviceUrl.defaultZone=http://discovery-server:8761/eureka/" \
 -e "spring.datasource.url=jdbc:mysql://mysql-master:3306/schooldb?useSSL=false&characterEncoding=UTF-8&serverTimezone=UTC&allowPublicKe
 -e "spring.datasource.username=school" \
 -e "spring.datasource.password=school" \
 class-server:1.0

# 집킨 주소로 변경
# mysqldb 컨테이너 띄운 이름으로 접근 가능 (추가로 뒤에 접근 권한 주어야 할듯)
  -e "logging.file=/api-logs/users-ws.log" \
```

▼ 11. Rent-server 배포

```
# Dockerfile
FROM amazoncorretto:17

VOLUME /tmp
```

```
COPY build/libs/rent-server-1.0.jar RentServer.jar

ENTRYPOINT ["java", "-jar", "RentServer.jar"]
```

```
# Docker 빌드
docker build -t rent-server:1.0 .

# Docker 배포
docker run -d --network jutopia-network \
 --name rent-server \
 -e "spring.cloud.config.uri=http://config-server:8888" \
 -e "spring.rabbitmq.host=rabbitmq" \
 -e "spring.zipkin.base-url=http://zipkin:9411" \
 -e "eureka.client.serviceUrl.defaultZone=http://discovery-server:8761/eureka/" \
 -e "spring.redis.host=redis" \
 -e "spring.redis.port=46379" \
 rent-server:1.0

# 집킨 주소로 변경
# mysqldb 컨테이너 띄운 이름으로 접근 가능 (추가로 뒤에 접근 권한 주어야 할듯)
  -e "logging.file=/api-logs/users-ws.log" \
```

▼ 12. Member-server 배포

```
# Dockerfile
FROM amazoncorretto:17

VOLUME /tmp

COPY build/libs/member-server-1.0.jar MemberServer.jar

ENTRYPOINT ["java", "-jar", "MemberServer.jar"]
```

```
# Docker 빌드
docker build -t member-server:1.0 .

# Docker 배포
docker run -d --network jutopia-network \
 --name member-server \
 -e "spring.cloud.config.uri=http://config-server:8888" \
 -e "spring.rabbitmq.host=rabbitmq" \
 -e "spring.zipkin.base-url=http://zipkin:9411" \
 -e "eureka.client.serviceUrl.defaultZone=http://discovery-server:8761/eureka/" \
 -e "spring.datasource.url=jdbc:mysql://mysql-master:3306/jutopia?useSSL=false&characterEncoding=UTF-8&serverTimezone=UTC&allowPublicKey
 -e "spring.datasource.username=root" \
 -e "spring.datasource.password=juto1234" \
 -e "spring.redis.host=redis" \
 -e "spring.redis.port=46379" \
 member-server:1.0

# 집킨 주소로 변경
# mysqldb 컨테이너 띄운 이름으로 접근 가능 (추가로 뒤에 접근 권한 주어야 할듯)
  -e "logging.file=/api-logs/users-ws.log" \
```

▼ 13. Stock-server 배포

```
# Dockerfile
FROM amazoncorretto:17

VOLUME /tmp

COPY build/libs/stock-server-1.0.jar StockServer.jar
```

```
ENTRYPOINT ["java", "-jar", "StockServer.jar"]
```

```
# Docker 빌드
docker build -t stock-server:1.0 .

# Docker 배포
docker run -d --network jutopia-network \
 --name stock-server \
 -e "spring.cloud.config.uri=http://config-server:8888" \
 -e "spring.rabbitmq.host=rabbitmq" \
 -e "spring.zipkin.base-url=http://zipkin:9411" \
 -e "eureka.client.serviceUrl.defaultZone=http://discovery-server:8761/eureka/" \
 stock-server:1.0

# 집킨 주소로 변경
# mysqldb 컨테이너 띄운 이름으로 접근 가능 (추가로 뒤에 접근 권한 주어야 할듯)
  -e "logging.file=/api-logs/users-ws.log" \
```

▼ 14. chat-server 배포

```
# Dockerfile
FROM python:3.11

COPY ./main.py ./main.py

COPY ./utils.py ./utils.py

COPY ./requirements.txt ./requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 9091

CMD ["uvicorn", "main:app", "--host=0.0.0.0", "--port=9092"]
```

```
# Docker 빌드
$ cd ~/S09P22C108/jutopia/chat-server/
$ sudo docker build -t chat-server:1.0 .

# Docker 배포
$ docker run -d --name chat-server -p 9002:9002 --network jutopia-network \
chat-server:1.0
위 명령어가 안되면 ps에서 돌아가고 있는것
```

▼ compose 디렉토리 docker-compose up 정리

하이픈 '-' 유무 정리:

`docker-compose` 로 돌리냐, `docker compose` 로 돌리냐

compose/db/mongo$ `docker-compose` up -d

compose/db/mysql$ `docker-compose` up -d

compose/db/redis$ `docker-compose` up -d

(정리 안된 경로는 아직 정리 안함. docker-compose 로 안되면 docker compose 로 돌리면 된다.)

▼ 도커 로그 수집 쉘 실행

```
이 Markdown 주석 밑에 작성해 놓은 명령어 만으로는 중간에 에러가 나서
종료됐을 시 재시작이 안되서
/etc/systemd/system/docker-log-sync.service 를 작성해서
꺼지면 항상 재시작되도록 했습니다.

파일내용:
  [Unit]
  Description=Docker Log Sync Service

  [Service]
  Type=simple
  ExecStart=/bin/bash /home/ubuntu/S09P22C108/jutopia/sync_logs.sh
  Restart=always
  User=root
  Group=root
  Environment=PATH=/usr/bin:/usr/local/bin

  [Install]
  WantedBy=multi-user.target

그외 밑 명령어 이용

sudo chmod 644 /etc/systemd/system/docker-log-sync.service
sudo systemctl daemon-reload
sudo systemctl start docker-log-sync.service
sudo systemctl status docker-log-sync.service
sudo systemctl enable docker-log-sync.service
echo 524288 | sudo tee /proc/sys/fs/inotify/max_user_watches
sudo systemctl restart docker-log-sync.service
```

실행 명령어

```
$ sudo nohup ~/S09P22C108/jutopia/sync_logs.sh &> /dev/null &
$ disown
```

스크립트 설명

- `sudo` : /var/lib/docker 에 접근하려면 sudo 권한 필요 (sync_logs.sh 파일 `chmod 755` 이상 해줘야함)
- `nohup` : 로그아웃으로 세션과 연결 종료되도 데몬 형태로 실행되어 종료되지 않고 계속 실행
- `~/S09P22C108/jutopia/sync_logs.sh` : sh 파일 실행
- `&> /dev/null` : 출력을 /dev/null 로 redirect (안해주면 터미널에 계속 로그 뜸)
- `&` : 백그라운드로 실행
- `disown` : 터미널과 프로세스 분리 (터미널 꺼져도 프로세스 계속 돌아감)

Dependency

inotifywait, rsync 설치 되있어야함

`$ sudo apt-get install inotifywait rsync`

실행된 sh kill하기

```
# 프로세스 ID 찾기
$ ps -ef | grep sync_logs.sh
```

```
# 찾은 PID kill
$ kill [PID]
```

sync_logs.sh 설명

```
#!/bin/bash
# 실행 안되면 sudo apt-get install inotifywait rsync
while true; do
    inotifywait -e modify /var/lib/docker/containers/*/*.log 2>>/tmp/inotify_errors.log
    if [ $? -ne 0 ]; then
        echo "inotifywait exited with error code $?" >> /tmp/inotify_errors.log
    fi
    rsync -av /var/lib/docker/containers/*/*.log /home/ubuntu/S09P22C108/jutopia/shared/logs/
done
```

`inotifywait` 가 **/var/lib/docker/containers/*/*.log** 파일들의 변경을 모니터링 하면 `rsync` 가 그 파일을 **/home/ubuntu/S09P22C108/jutopia/shared/logs/** 디렉토리로 복사

▼ 젠킨스 배포

```
if [ ${PROJECT_NAME} = "config-server" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/config-server
  chmod +x gradlew
  ./gradlew clean build
  echo "config-server docker start"
  docker stop config-server || true
  docker rm config-server || true
  docker rmi config-server:1.0 || true
  docker build -t config-server:1.0 .
  docker run -d -p 8888:8888 --network jutopia-network -e "spring.rabbitmq.host=rabbitmq" -e "spring.profiles.active=default" --name cor

# elif [ ${PROJECT_NAME} = "discovery-server" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/discovery-server
  chmod +x gradlew
  ./gradlew clean build
  echo "discovery-server docker start"
  docker stop discovery-server || true
  docker rm discovery-server || true
  docker rmi discovery-server:1.0 || true
  docker build -t discovery-server:1.0 .
  docker run -d -p 8761:8761 --network jutopia-network -e "spring.cloud.config.uri=http://config-server:8888" --name discovery-server di

# elif [ ${PROJECT_NAME} = "apigateway-server" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/apigateway-server
  chmod +x gradlew
  ./gradlew clean build -x test
  echo "apigateway-server docker start"
  docker stop apigateway-server || true
  docker rm apigateway-server || true
  docker rmi apigateway-server:1.0 || true
  docker build -t apigateway-server:1.0 .
  docker run -d -p 8000:8000 --network jutopia-network -e "spring.cloud.config.uri=http://config-server:8888" -e "spring.rabbitmq.host=r

#elif [ ${PROJECT_NAME} = "class-server" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/class-server
  chmod +x gradlew
  ./gradlew clean build -x test
  echo "class-server docker start"
  docker stop class-server || true
  docker rm class-server || true
  docker rmi class-server:1.0 || true
  docker build -t class-server:1.0 .
  docker run -d --network jutopia-network --name class-server -e "spring.cloud.config.uri=http://config-server:8888" -e "spring.rabbitmq

#elif [ ${PROJECT_NAME} = "rent-server" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/rent-server
  chmod +x gradlew
  ./gradlew clean build -x test
  echo "rent-server docker start"
  docker stop rent-server || true
```

```
  docker rm rent-server || true
  docker rmi rent-server:1.0 || true
  docker build -t rent-server:1.0 .
  docker run -d --network jutopia-network --name rent-server -e "spring.cloud.config.uri=http://config-server:8888" -e "spring.rabbitmq.

#elif [ ${PROJECT_NAME} = "news-server" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/news-server
  echo "news-server docker start"
  docker stop news-server || true
  docker rm news-server || true
  docker rmi news-server:1.0 || true
  docker build -t news-server:1.0 .
  docker run -d --name news-server -p 9001:9001 --network jutopia-network news-server:1.0

#elif [ ${PROJECT_NAME} = "member-server" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/member-server
  chmod +x gradlew
  ./gradlew clean build -x test
  echo "member-server docker start"
  docker stop member-server || true
  docker rm member-server || true
  docker rmi member-server:1.0 || true
  docker build -t member-server:1.0 .
  docker run -d --network jutopia-network --name member-server -e "spring.cloud.config.uri=http://config-server:8888" -e "spring.rabbitm

#elif [ ${PROJECT_NAME} = "stock-server" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/stock-server
  chmod +x gradlew
  ./gradlew clean build -x test
  echo "stock-server docker start"
  docker stop stock-server || true
  docker rm stock-server || true
  docker rmi stock-server:1.0 || true
  docker build -t stock-server:1.0 .
  docker run -d --network jutopia-network --name stock-server -e "spring.cloud.config.uri=http://config-server:8888" -e "spring.rabbitmq

# elif [ ${PROJECT_NAME} = "chat-server" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/chat-server
  echo "chat-server docker start"
  docker stop chat-server || true
  docker rm chat-server || true
  docker rmi chat-server:1.0 || true
  docker build -t chat-server:1.0 .
  docker run -d --name chat-server -p 9002:9002 --network jutopia-network chat-server:1.0

#elif [ ${PROJECT_NAME} = "teacher" ]; then
  cd /var/jenkins_home/workspace/jutopia_project/jutopia/teacher
  chmod +x gradlew
  ./gradlew clean build -x test
  echo "teacher docker start"
  docker stop teacher || true
  docker rm teacher || true
  docker rmi teacher:1.0 || true
  docker build -t teacher:1.0 .
  docker run -d --network jutopia-network --name teacher -e "eureka.client.serviceUrl.defaultZone=http://discovery-server:8761/eureka/"
fi
```