Apache Kafka
생산자 및 소비자 API 사용

Intro

Producer-Consumer 예제

- 예제 애플리케이션은 Producer-Consumer 하위 디렉터리의 https://github.com/Azure-Samples/hdinsight-kafka-java-get-started 에 존재
- 애플리케이션은 주로 4개의 파일로 구성됨
- AdminClientWrapper.java: 관리자 API를 이용하여 토픽의 생성, 설명 및 삭제를 담당함
- Producer.java: 생산자 API를 사용하여 Kafka에 임의의 문장을 보냄
- Consumer.java: 소비자 API를 사용하여 Kafka에서 데이터를 읽고 STDOUT에 내보냄
- Run.java: 생산자 및 소비자 코드를 실행하는 데 사용되는 명령줄 인터페이스

Run.java

```
// 프로듀서 및 컨슈머 핸들링 시작
public class Run {
    public static void main(String[] args) throws IOException {
switch(args[0].toLowerCase()) {
            case "producer":
                Producer.produce(brokers, topicName);
                break;
            case "consumer":
           …..
                Consumer.consume(brokers, groupId, topicName);
                break;
            case "describe":
                AdminClientWrapper.describeTopics(brokers, topicName);
                break;
            case "create":
           …..
                AdminClientWrapper.createTopics(brokers, topicName, partition, replication);
                break;
            case "delete":
                AdminClientWrapper.deleteTopics(brokers, topicName);
                break;
            default:
                usage();
        }
        System.exit(0);
    }
…
}
```

Producer.java

```
public class Producer
{
    public static void produce(String brokers, String topicName) throws IOException
    {
        // producer 사용을 위해 properties 설정
        Properties properties = new Properties();
        // 브로커 서버 설정
        properties.setProperty("bootstrap.servers", brokers);
        // 키(key)/값(value) 쌍을 어떻게 직렬화 할지 설정
        properties.setProperty("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
      properties.setProperty("value.serializer","org.apache.kafka.common.serialization.StringSerializer");
        // 설정된 properties를 이용해 producer client 생성
        KafkaProducer<String, String> producer = new KafkaProducer<>(properties);
         ….
        // 여러 개의 레코드를 보냄(produce함)
        for(int I = 0; I < 100; i++) {
            // 랜덤한 문장을 뽑음
            String sentence = sentences[random.nextInt(sentences.length)];
            // 지정된 토픽으로 문장이 producer client를 통해 전송
            try
            {
                producer.send(new ProducerRecord<String, String>(topicName, sentence)).get();
            }
            catch (Exception ex)
            {
                System.out.print(ex.getMessage());
                throw new IOException(ex.toString());
            }
           …
        }
        …
    }
}
```

Consumer.java

```java
public class Consumer {
    public static int consume(String brokers, String groupId, String topicName) {
        // consumer 생성
        KafkaConsumer<String, String> consumer;
        // consumer를 설정 (Configure)
        Properties properties = new Properties();
        // Kafka 클러스터의 브로커를 잡아줌
        properties.setProperty("bootstrap.servers", brokers);
        // Consumer 그룹을 설정 (모든 consumer들은 설정된 그룹에 속해 있어야함).
        properties.setProperty("group.id", groupId);
        ……
        // 그룹이 이미 있던 그룹이 아니라 새로 생성된 그룹이면, 어디서 부터 읽어야 하는 지의 offset 데이터가 없음
        // 그렇기 때문에 데이터 스트림의 가장 첫번째 스트림 부터 읽어야 한다고 명시해줌
        properties.setProperty("auto.offset.reset","earliest");

        consumer = new KafkaConsumer<>(properties);

        // 지정된 토픽을 구독(subscribe)함
        consumer.subscribe(Arrays.asList(topicName));

        // ctrl + c 누를 때까지 레코드를 받음
        int count = 0;
        while(true) {
            // 레코드를 가져옴(poll)
            ConsumerRecords<String, String> records = consumer.poll(200);
            // 실제로 가져온 레코드의 크기가 존재하는지 확인
            if (records.count() == 0) {
                // 타임아웃 / 읽을 것이 없음
            } else {
                for(ConsumerRecord<String, String> record: records) {
                    count += 1;
                    System.out.println( count + ": " + record.value()); // 레코드와 그에 해당되는 번호를 출력
                }
            }
        }
```

AdminClientWrapper.java

```
public class AdminClientWrapper {
        ……
    public static void describeTopics(String brokers, String topicName) throws IOException {
        // 관리자 client 사용을 위해 properties 설정
        Properties properties = getProperties(brokers);

        try (final AdminClient adminClient = KafkaAdminClient.create(properties)) {
            final DescribeTopicsResult describeTopicsResult =
                      adminClient.describeTopics(Collections.singleton(topicName));
            TopicDescription description = describeTopicsResult.values().get(topicName).get();
            System.out.print(description.toString());
        } ……
    }
    public static void deleteTopics(String brokers, String topicName) throws IOException {
        // 관리자 client 사용을 위해 properties 설정
        Properties properties = getProperties(brokers);

        try (final AdminClient adminClient = KafkaAdminClient.create(properties)) {
            final DeleteTopicsResult deleteTopicsResult =
                      adminClient.deleteTopics(Collections.singleton(topicName));
            deleteTopicsResult.values().get(topicName).get();
            System.out.print("Topic " + topicName + " deleted");
        } …….
    }
    public static void createTopics(String brokers, String topicName ….) throws IOException {
        // 관리자 client 사용을 위해 properties 설정
        Properties properties = getProperties(brokers);

        try (final AdminClient adminClient = KafkaAdminClient.create(properties)) {
            final NewTopic newTopic = new NewTopic(topicName, numPartitions, replicationFactor);

            final CreateTopicsResult createTopicsResult =
                      adminClient.createTopics(Collections.singleton(newTopic));
            createTopicsResult.values().get(topicName).get();
            System.out.print("Topic " + topicName + " created");
        } ……
```