Kafka Streams: Interactive Queries

Lee Dongjin (dongjin@apache.org)

오늘 이야기할 주제는…

- “Kafka Topic에 저장된 내용을 Microservice에서 참조하고 싶은데, 어떻게 해야 할지 모르겠어요.”
- “Redis 같은 데 저장해 놓고 써야 하나요?”
  - Producer → Kafka → Consumer → Redis → Microservice?
  - 좀 더 쉬운 방법이…?
- 결론부터 이야기하자면: 방법이 있습니다.
  - Kafka Streams의 Interactive Query 기능 (혹은, “Queryable Store” 기능)

Kafka Streams: 초간단 소개

- Kafka 0.10에서부터 도입된 Stream 처리 library
  - Kafka Topic을 실시간으로 받아서 처리하는 루틴을 간편하게 정의할 수 있음.
    - High-Level DSL: KStream, KTable, …
    - Low-Level API도 지원. (숙련자용)
  - 장점
    - Task 관리가 필요 없음.
      - 필요한 만큼 (=partition 수) 알아서 작업(task)을 생성하고 thread pool에 분배함.
      - 하나의 작업이 하나의 (topic, partition)에 대한 처리를 전담.
    - Kafka의 consumer group 기능을 사용해서 구현됨.
      - 별도의 coordination이 필요 없음 - 효율적.
    - Framework (X) Library (O): 어디에서나 그냥 불러서 쓰면 됨.

Kafka Streams: wordcount (1)

```
// Build Topology with StreamsBuilder
final StreamsBuilder builder = new StreamsBuilder();

// KStream: unbounded series of records
final KStream<String, String> source = builder.stream(inputTopic);

// Transform input records into stream of words with `flatMapValues` method
final KStream<String, String> tokenized = source
  .flatMapValues(value ->
Arrays.asList(value.toLowerCase(Locale.getDefault()).split(" ")));
```

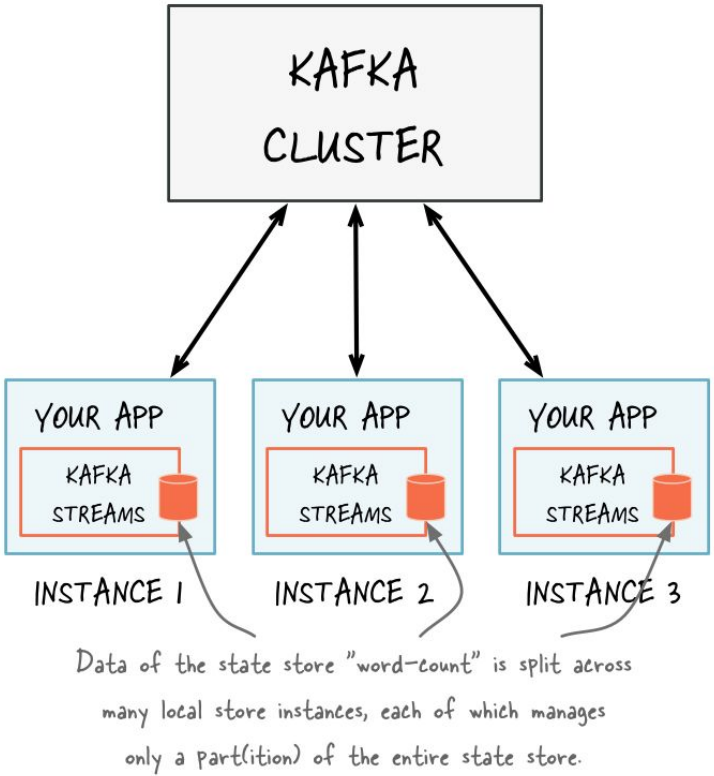
Kafka Streams: wordcount (2)

```
// KTable: Stateful abstraction of aggregated stream
// Build KTable from KStream by group and aggregate operations
final KTable<String, Long> counts = tokenized.groupBy((key, value) ->
value).count();

// Write back aggregated status to output kafka topic
counts.to(outputTopic, Produced.with(Serdes.String(), Serdes.Long()));

// Build Topology instance
return builder.build();
```

Kafka Streams: wordcount (3)

```
public static void main() {
  // Build Topology with StreamsBuilder
  Properties props = ... // Configuration properties
  Topology topology = ... // Topology object

  final KafkaStreams streams = new KafkaStreams(topology, props);

  /* Omit some boilerplate codes... */
  // Start the Kafka Streams application
  streams.start();
}
```

Kafka Streams: wordcount (3)



Interactive Query

- Key-value storage
  - Kafka Streams가 처리의 효율성을 위해 내부적으로 생성하는 저장소.
  - RocksDB로 구현됨. (기본값)
  - 직접 생성할 수도 있고, 사용자가 임의로 생성할 수도 있음.
  - KTable 객체를 생성하면 반드시 하나가 함께 생김.
- Interactive Query
  - KafkaStreams 내부에 생성된 key-value storage의 내용과 위치를 조회할 수 있도록 해 주는 기능.
  - “Queryable Store”: interactive query 기능이 설정된 key-value storage.
  - ‘수정’ 은 안 됨.

Interactive Query: example (1)

Before:

```
final KTable<String, Long> counts = tokenized
                                    .groupBy((key, value) -> value)
                                    .count();
```

After:

```
final KTable<String, Long> counts =
  tokenized.groupBy((key, value) -> value)
           .count(
             Materialized.<String, Long, KeyValueStore<Bytes, byte[]>
               as("word-count")
           )); // Configure queryable store named ‘word-count’
```

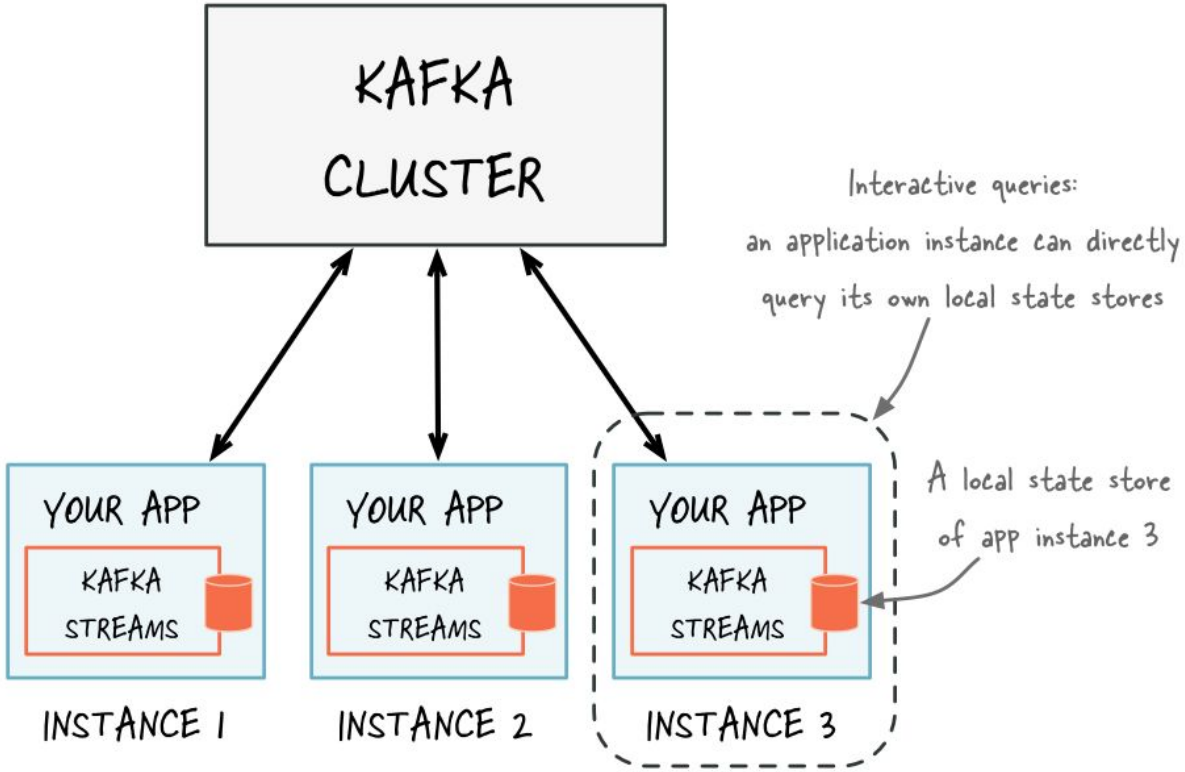
Interactive Query: example (2)

```
// props: StreamsConfig.APPLICATION_SERVER_CONFIG -> {host:port}
final KafkaStreams streams = new KafkaStreams(topology, props);
streams.start();

// Get access to the store
ReadOnlyKeyValueStore<String, Long> store =
  streams.store("word-count", QueryableStoreTypes.keyValueStore());

// Query the store
Long count = store.get("kafka");
```

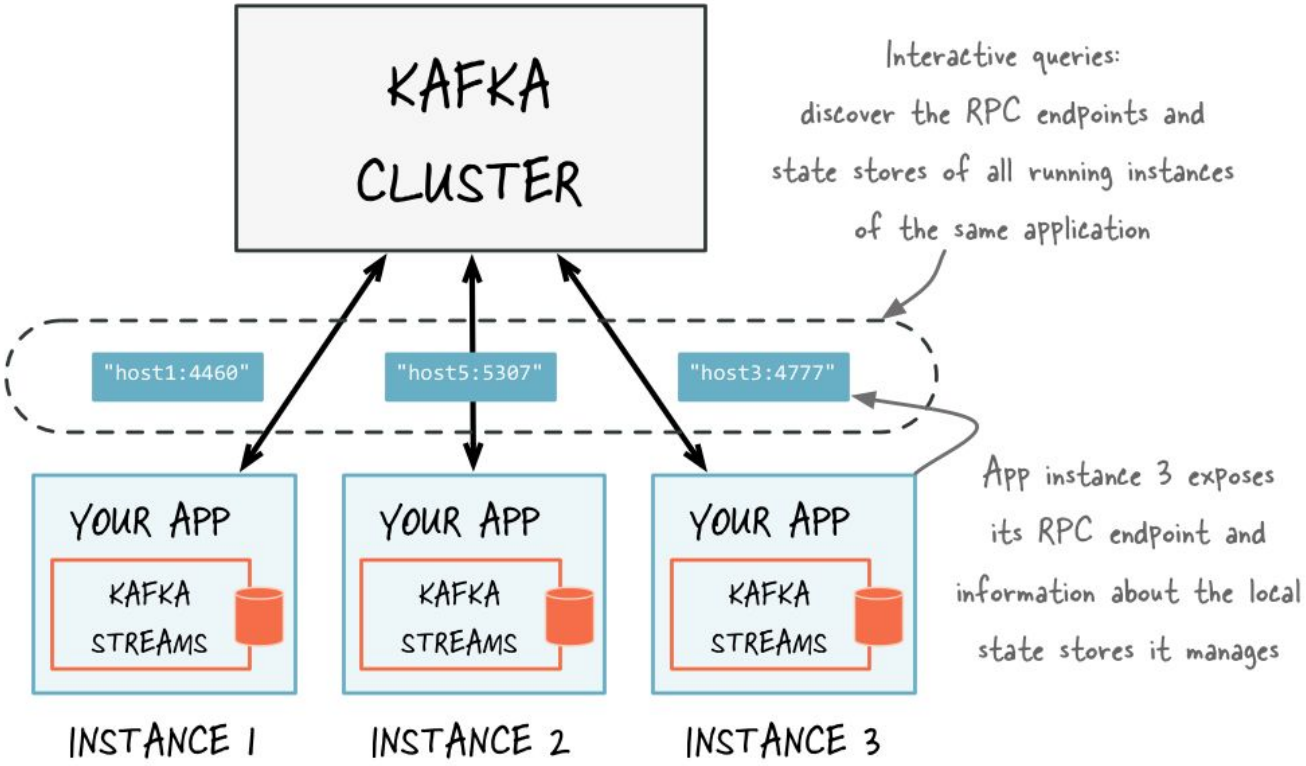
Interactive Query: example (3)

```
// Find all the locations of instances
// with the state store named "word-count"
Collection<StreamsMetadata> wordCountHosts =
  streams.allMetadataForStore("word-count");

// Find the location of instance with the state store with "kafka"
StreamsMetadata metadata = streams
      .metadataForKey("word-count", "kafka", Serdes.String().serializer());
```

Interactive Query: example (4)



Interactive Query: example (5)



결론

- KTable을 사용하면 Kafka Topic의 내용을 표 형태로 읽어올 수 있다.
  - 그리고 그 내용을 Interactive Query를 사용해서 열어볼 수 있다.
- Kafka Streams가 해 주는 것:
  - 현재 process에서 잡고 있는 partition에 포함된 key에 대한 value값.
  - 현재 process에서 잡고 있는 partition에 포함되지 않은 key가 저장되어 있는 위치. (host:port)
- 직접 해 줘야 하는 것:
  - Remote API Protocol 선택: HTTP, gRPC, ...
  - Remote API 구현
    - (해당 instance가 보유한) '주어진 key에 해당하는 값을 리턴하는 api'
    - '주어진 key에 해당하는 값이 어디 있는지를 리턴하는 api'

감사합니다!

- 질문?