Apache Kafka

Agenda

- Apache Kafka overview
- Apache Kafka applications
- Spring Kafka framework

Apache Kafka overview

Apache Kafka overview

- Apache Kafka is a distributed streaming platform
- More performant than RabbitMQ in many scenarios
- Also provides more features than RabbitMQ
- However it is more complex for configuration and the learning curve is steeper !

It is not uncommon that some companies that use Apache Kafka heavily (like LinkedIn) have dedicated teams responsible for running and administering Apache Kafka instances !

Notorious companies using Apache Kafka include also Yahoo, Uber, Airbnb, Spotify, Twitter, Paypal and Netflix among others.

Apache Kafka overview

- Apache Kafka aims to be a high-throughput, low-latency platform for real-time data feeds

- Messages processed by Kafka are immutable (i.e. cannot be modified once placed in the Kafka instance)

Throughput: the amount of data the system can process over a period of time
Latency: the time required for the system to process a single request

Kafka record

- The message processed by Apache Kafka is called a **record** and consists of three parts:
  - key: used by clients to identify the message
  - value: the actual payload of the message
  - timestamp: set when message is received by the broker
- Streams of records are stored in **topics** in Apache Kafka

Information about Kafka topics is stored in a distributed manner in Apache Zookeeper

Kafka topics

- Kafka topics are of two types:
  - **delete**: deletes messages according to certain criteria such as size of the topic or time of the message
  - **compaction**: performs an upsert, i.e. if a message arrives with a key already present in the topic the old message gets updated, otherwise a new message is inserted
- A topic spans over the entire Kafka cluster, not only in a single Apache Kafka instance

Kafka partitions

- Since topics span a Kafka cluster each topic can be divided into partition
- A partition represents the portion of messages stored on the particular Kafka instance from the Kafka cluster
- Partitioning of topics enables scalability and high availability in Apache Kafka

Kafka partitions

- The number of partitions determines the scalability of a particular topic
- Number of partitions are configurable
- The producer uses a partitioning scheme to split messages across partitions

Kafka subscribers

- Multiple subscribers can subscribe and receive messages from a Kafka topic
- A single subscriber can receive messages asynchronously from more than a single topic
- Subscribers can be combined in subscriber groups for the purpose of subscriber failover (just one subscriber from group reads from a partition)

Kafka scalability

- Apache Kafka is aimed to scale horizontally
- Uses Apache Zookeeper for distributed coordination

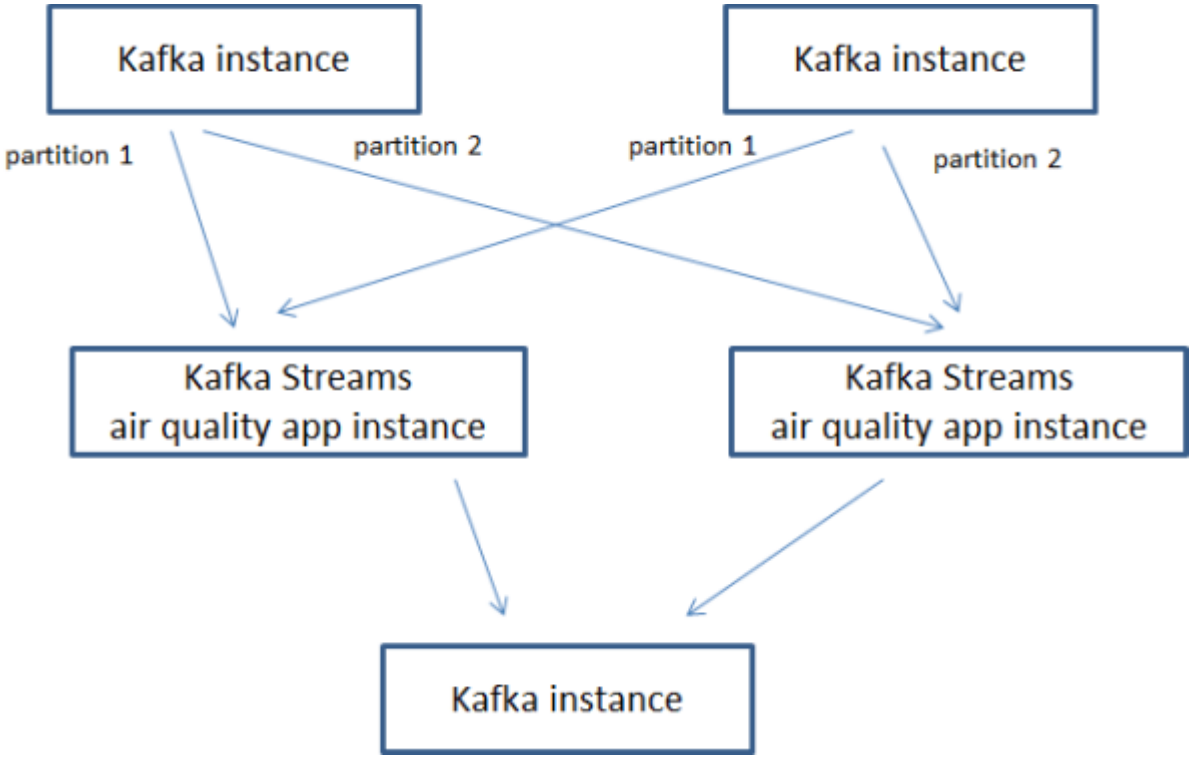
Kafka reliability

- Reliability is achieved by defining a replication factor at the topic level
- The replication factor determines how many times a topic should be replicated across the instances in the Kafka cluster
- The replication factor takes place at the partition level

Kafka APIs

- Apart from the producer/consumer APIs Kafka also provides:
  - Kafka Streams library for building streaming application on top of Apache Kafka
  - Kafka Connect API for data import/export to/from Kafka topics

Kafka Streams



Apache Kafka applications

Kafka Java client

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.5.0</version>
</dependency>
```

Kafka publisher

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("acks", "all");
props.put("retries", 0);
props.put("batch.size", 16384);
props.put("linger.ms", 1);
props.put("buffer.memory", 33554432);
props.put("key.serializer",
  "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer",
  "org.apache.kafka.common.serialization.StringSerializer");
Producer<String, String> producer =
       new KafkaProducer<>(props);

…
producer.send(new ProducerRecord<String,String>(“topic",
       record));
```

Kafka publisher (transactional)

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("transactional.id", "my-transactional-id");
Producer<String, String> producer =
       new KafkaProducer<>(props, new StringSerializer(),
       new StringSerializer());
producer.initTransactions();
try {
     producer.beginTransaction();
     producer.send(…); …; producer.send(…);
     producer.commitTransaction();
} catch (ProducerFencedException
       | OutOfOrderSequenceException
       | AuthorizationException e) {
       producer.close();
} catch (KafkaException e) {
       producer.abortTransaction();
}
producer.close();
```

Kafka consumer

```
Properties props = new Properties();
props.setProperty("bootstrap.servers", "localhost:9092");
props.setProperty("group.id", "test");
props.setProperty("enable.auto.commit", "true");
props.setProperty("auto.commit.interval.ms", "1000");
props.setProperty("key.deserializer",
org.apache.kafka.common.serialization.StringDeserializer");
props.setProperty("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
KafkaConsumer<String, String> consumer =
      new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("topic1", "topic2"));
while (true) {
      ConsumerRecords<String, String> records =
            consumer.poll(Duration.ofMillis(100));
      for (ConsumerRecord<String, String> record : records)
            …
}
```

Kafka Streams

```
List result = new LinkedList();
final Serde stringSerde = Serdes.String();
final StreamsBuilder builder = new StreamsBuilder();
builder.stream("numbers", Consumed.with(stringSerde,
stringSerde))
       .map((key, value) -> new KeyValue(key,
Integer.valueOf(value))).
                    filter((key, value) -> value >
THRESHOLD)
       .foreach((key, value) -> {
                    result.add(value.toString());
                });

final Topology topology = builder.build();
final KafkaStreams streams = new KafkaStreams(topology,
                createKafkaStreamsConfiguration());
streams.start();
…
streams.close();
```

Kafka Streams

```
private Properties createKafkaStreamsConfiguration() {

        Properties props = new Properties();
        props.put(StreamsConfig.APPLICATION_ID_CONFIG,
                "text-search-config");
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
                "localhost:9092");
        props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
                Serdes.String().getClass());
        props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
                Serdes.String().getClass());
        return props;
}
```

Spring Kafka framework

Spring Kafka framework

```
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
  <version>2.5.4.RELEASE</version>
</dependency>
```

Spring kafka framework

- Spring Kafka framework provides convenient utilities for integrating Kafka within a Spring application
- Support is similar to the one provided for RabbitMQ by the Spring RabbitMQ framework

Spring kafka framework

- Spring Kafka framework provides:
  - KafkaTemplate for publishing messages
  - KafkaAdmin for configuring topics and other Kafka items
  - Kafka listeners (via @KafkaListener) for creating listeners to Kafka topics (bean methods that receive the records)
  - a number of Spring events related to Kafka publishing/subscription

Spring boot

- Spring boot provides autoconfiguration of the Spring Kafka project

- The following properties can be specified in **application.properties**:
  - spring.kafka.bootstrap-servers=localhost:9092
  - spring.kafka.consumer.group-id=<groupId>

Thanks

?