Kafka and Streaming

SciSpike

Introduction

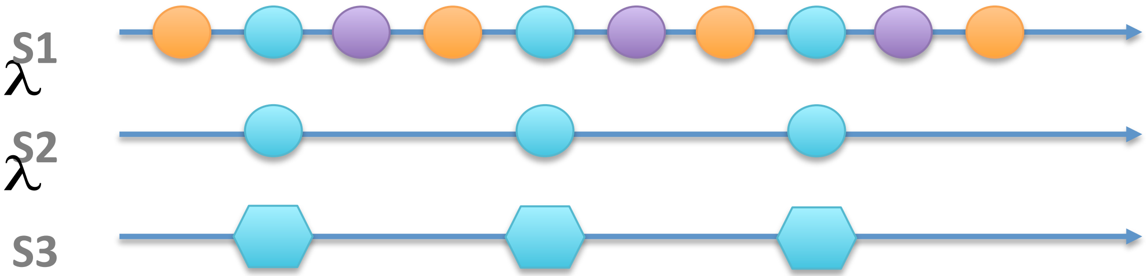
- Kafka Streaming: An alternative for streaming to and from Kafka
  - Part of the Apache Kafka
  - Powerful
    - Highly scalable, fault-tolerant
    - Rich feature set with support for both stateless and stateful processing
    - Support for fault-tolerant local state
  - Lightweight
    - Just a library integrated in Kafka (no external dependencies)
    - No need for dedicated clusters
  - Near real-time
    - Millisecond processing latency
- Kafka and Spark / Flink

Two API's

- Kafka supports a low level API that can be used to manipulate streams

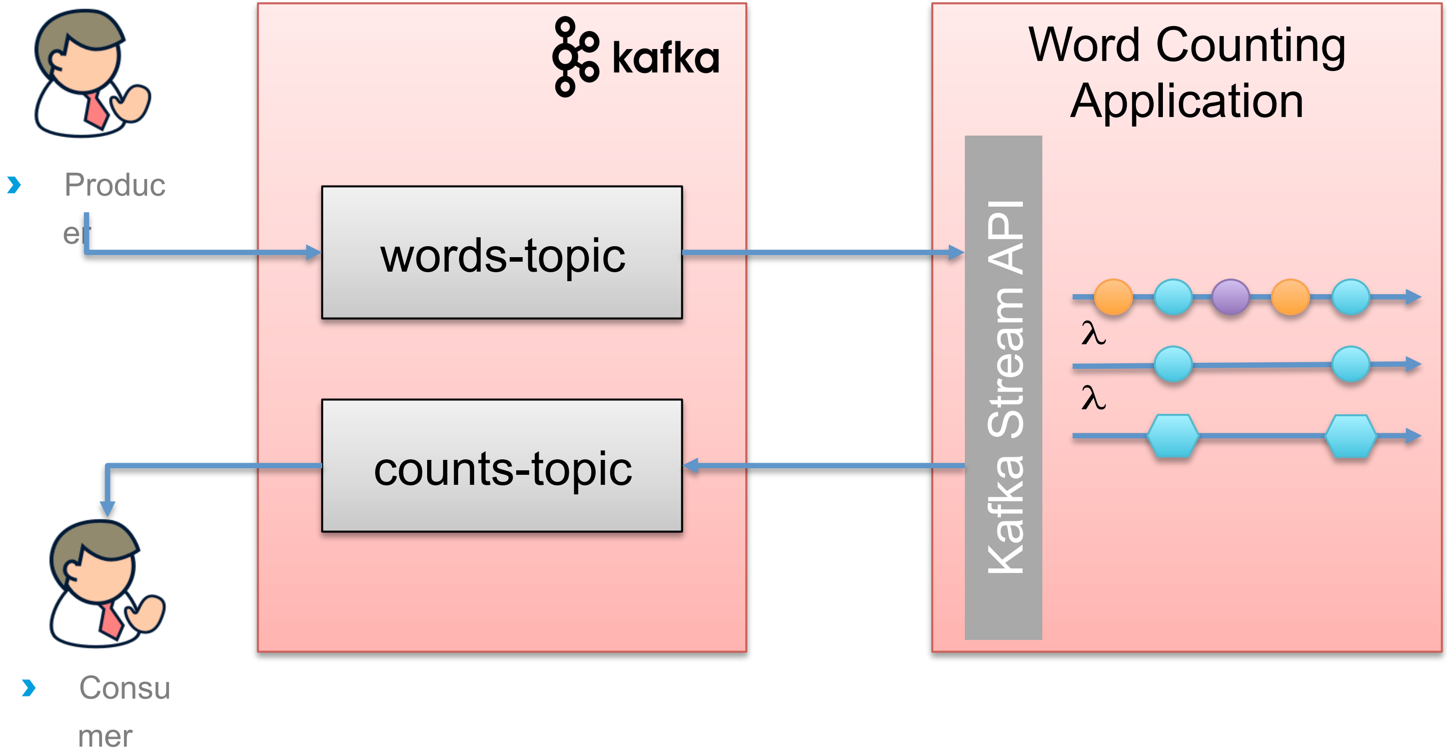
- We'll focus on the Kafka Stream `DSL`
  - Higher level
  - Closer to other streaming ideas
    - Brining in some of the functional programming paradigms

What is Stream Processing?



- Programming with streams of data (even unbounded streams)
- Typical traits
  - Functional
    - Apply a dataflow or a sequence of functions to the streams
    - Combining functions provides a powerful expression language (lambda expression)
  - Reactive
    - Process incoming stream data immediately upon receipt

The Obligatory Word Count Example



- Producer

kafka

words-topic

counts-topic

Word Counting Application

Kafka Stream API

λ

λ

- Consumer

Kafka Stream Classes

- StreamsConfig
  - A wrapper around a map used for configuration
- KStreamBuilder
  - The key class for setting up the stream procoessing
  - Based on the GoF Builder Pattern
    - Fluent API
    - Constructs the processing dataflow
- KafkaStreams
  - A wrapper of the actual stream

Enable Kafka Stream (Using Maven)

```
<dependency>
	<groupId>org.apache.kafka</groupId>
	<artifactId>kafka-streams</artifactId>
	<version>0.10.1.1</version>
</dependency>
```

- Kafka Streams are distributed as a simple jar file
- Simply include the jar file in your program and you'll be able to use the streaming library

Programming to Stream

```
// Configure
Properties props = new Properties();
pros.put(StreamsConfig.CLIENT_ID_CONFIG, "...")
...
StreamsConfig config = new StreamsConfig(props);

// Serialization/deserialization
Serde<String> serDeser = Serdes.String();

// Create the data processing pipeline
KStreamBuilder bld = new KStreamBuilder();
bld.stream(serDeser, serDeser, "topic")
.map(...) ...

// Create stream
KafkaStreams sp = new KafkaStreams(bld, config);

// Start stream
kafkaStreams.start();
```

Word Count Streaming

```
StreamsConfig config = new StreamsConfig(props);
Serde<String> serde = Serdes.String();

KStreamBuilder bld = new KStreamBuilder();
bld.stream(serde, serde, "words-topic")
  .flatMapValues(text -> asList(text.split(" ")))
  .map((key, word) -> new KeyValue<>(word, word))
  .countByKey(serde, "Counts")
  .toStream()
  .map((word, count) ->
        new KeyValue<>(word, word + ":" + count))
  .to(serde, serde, "counts-topic");

KafkaStreams s= new KafkaStreams(bld, config);
s.start();
```

kafkacat – A Useful Tool

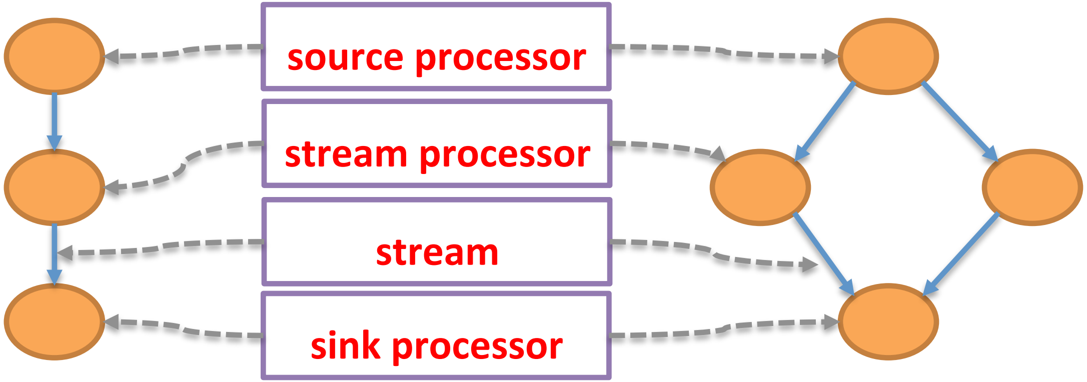
- kafkacat is a tool created by Magnus Edenhill
  - https://github.com/edenhill/kafkacat
  - Doesn't require JVM
  - Similar to 'netcat'
  - Can act as a Producer and consumer
  - Useful also to query metadata from Kafka
- To provide the word list to the application, we could simply use kafkacat instead of writing new producers and consumers
- Use as a producer
  - kafkacat –P –b myBroker –t topic1
- Use as a consumer
  - kafkacat –b myBroker –G mygroup topic1 topic 2

Installing kafkacat

- The installation of kafkacat is trivial on Mac most Linux distrobutions
  - Mac
    - brew install kafkacat
  - Ubuntu or Debian
    - [sudo] apt-get install kafkacat

- Our Kafka docker image is using Alpine distribution so if you want to run it in the docker image you'll have to:
  - apk add --update alpine-sdk bash python cmake
  - curl https://codeload.github.com/edenhill/kafkacat/tar.gz/master | tar xzf - && cd kafkacat-* && bash ./bootstrap.sh

Processor Topology



- The topology defines the computational logic of the data processing pipeline
- The topology typically forms a chain, but more advanced computation may be defined as a graph

Processing and Time

- In streaming time is an important design consideration
  - When working with concepts such as windowing, it is essential to establish which time to use
- Event-time
  - The point in time when an event or data record was created by the source
- Processing-time
  - The point in time when an event was processed by the stream processor
- Ingestion-time
  - The point in time when an event was stored in a topic partition by one of the Kafka brokers

Stateful Stream Processing

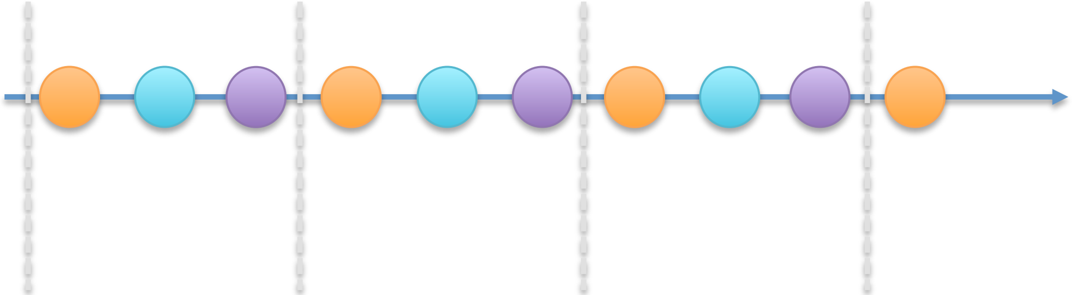
- Typically we try to make our stream processing stateless

- However, for some use cases, stateful streaming may be essential
- The Kafka Stream DSL provides support for stateful processing

Stream vs. Table

- Kafka streams introduces the concept of tables, where
  - A stream can be viewed as a table: KStream → KTable
  - A table can be viewed as a stream: KTable → KStream
  - (called stream-table duality)
- Streams as tables
  - A stream can be considered a changelog of a table
  - One can convert the stream into a table by replaying the event stream
- Table as streams
  - A table can be be converted into a stream by iterating over each key-value entry

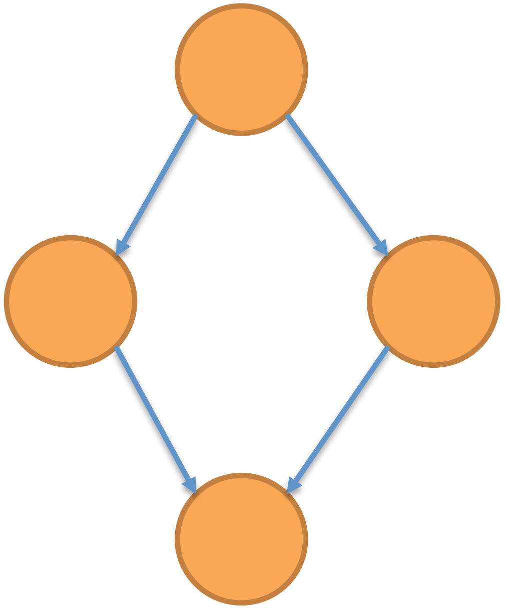
Windowing



- We may have to process data in time buckets, called windows
- Typically as part of some form of aggregation
- Windowing operations are available in the Kafka Stream DSL
- A window has a retention period to handle possible late arriving events

Parallel Stream Processing Pipelines

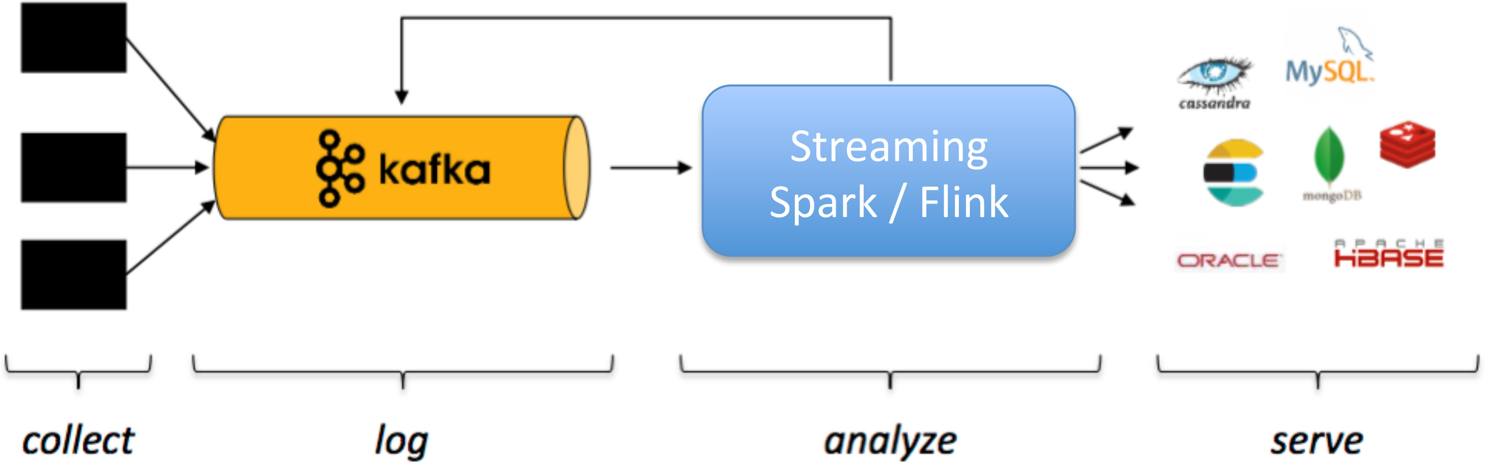
- The Kafka Stream DSL supports ways to parallelize and rendezvous processing steps
- Join
  - This operation merges two streams into a single stream
- Aggregation
  - Takes one input stream and yields a new stream by combining multiple input events into a single output event



Backpressure

- Kafka Streams don't need a backpressure mechanism
- Kafka uses a depth-first processing strategy
  - Each event consumed from Kafka goes through the complete processor topology before the next message is processed

Kafka with Spark or Flink



- Gather and backup streams
- Offer streams for consumption
- Provide stream recovery

- Analyze and correlate streams
- Create derived streams and state
- Provide these to downstream systems

Spark Streaming and Kafka

- We can set up Spark to use Kafka stream as the source of events

- Code structure:
  - A bit of configuration for Spark Streaming
  - A bit of configuration for Kafka parameters
  - Create stream from Kafka

- At this point, we have a DStream and we can do the normal Spark Streaming processing on it

Spark Streaming with Kafka – Spark Configuration

```
// Consume command line parameters
val Array(brokers, topics, interval) = args

// Create Spark configuration
val sparkConf = new SparkConf().setAppName("SparkKafka")

// Create streaming context, with batch duration in ms
val ssc = new StreamingContext(sparkConf, Duration(interval.toLong))
ssc.checkpoint("./output")
```

Spark Streaming with Kafka – Kafka Configuration

```
// Create a set of topics from a string
val topicsSet = topics.split(",").toSet

// Define Kafka parameters
val kafkaParams = Map[String, Object](
  "bootstrap.servers" -> brokers,
  "key.deserializer" -> classOf[StringDeserializer],
  "value.deserializer" -> classOf[StringDeserializer],
  "group.id" -> "use_a_separate_group_id_for_each_stream",
  "auto.offset.reset" -> "latest",
  "enable.auto.commit" -> (false: java.lang.Boolean))
```

Create a Kafka Stream in Spark and Use It

```
// Create a Kafka stream
val stream = KafkaUtils.createDirectStream[String, String](
  ssc, PreferConsistent, Subscribe[String, String](topicsSet,kafkaParams))
// Get messages - lines of text from Kafka
val lines = stream.map(consumerRecord => consumerRecord.value)
// Split lines into words
val words = lines.flatMap(_.split(" "))
// Map every word to a tuple
val wordMap = words.map(word => (word, 1))
// Count occurrences of each word
val wordCount = wordMap.reduceByKey(_ + _)
//Print the word count
wordCount.print()
```

Summary

- Kafka provides a stream processing library
- Two APIs
  - Low level API
  - Kafka Stream DSL
- The Kafka Stream DSL enables the definition of processing topologies
- Kafka streams support stateful stream processing
- Kafka has a reach support of functions and features
  - Windowing
  - Aggregation
  - Join
  - Functions (map, flatMap, etc.)
- Integration with other streaming systems: Spark, Flink, …