Apache Kafka

ECE 454 / 751: Distributed Computing

Instructor: Dr. Wojciech Golab

wgolab@uwaterloo.ca

Slides are derived from online materials, including the Apache Kafka website:
https://kafka.apache.org/

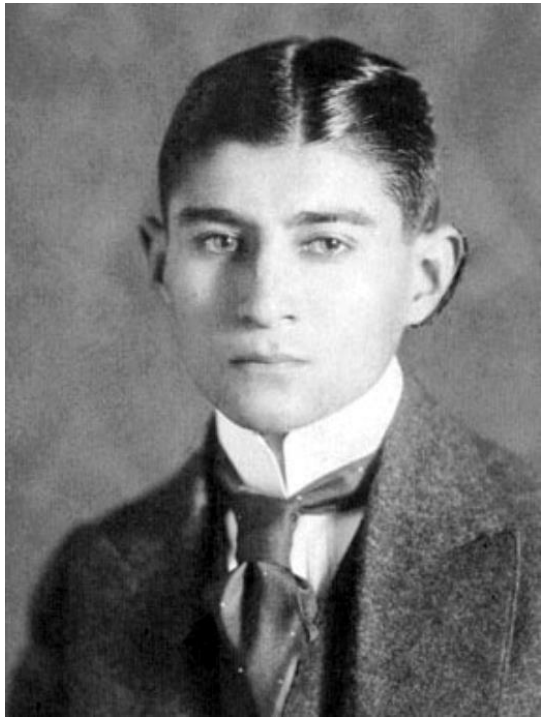
Learning objectives

To develop a working knowledge of Apache Kafka.

- clients and brokers
- producer API
- consumer API
- streams API

The name …

Franz Kafka
novelist and short story writer
1883–1924



…
As Gregor Samsa one morning from restless dreams awoke, found he himself in his bed into an enormous vermin transformed.
…

Image source:
https://en.wikipedia.org/wiki/Franz_Kafka#/media/File:Franz_Kafka_1910.jpg

Main features of Kafka

- Publish-subscribe (message-oriented communication).
- Real-time stream processing.
- Distributed and replicated storage of messages and streams.

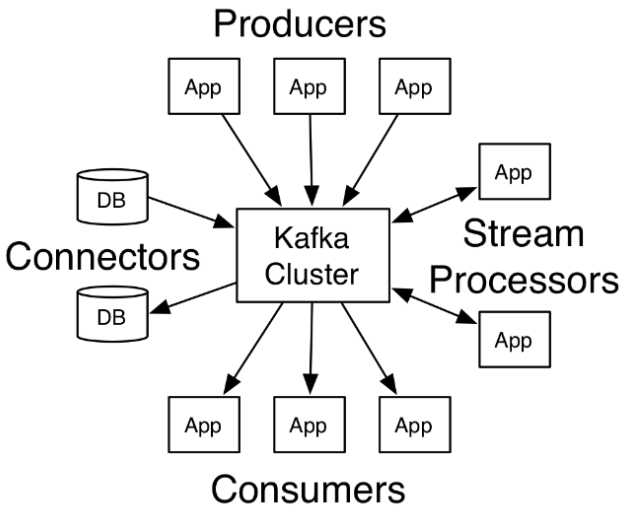


Image source: https://kafka.apache.org/25/images/kafka-apis.png

Use cases
(source: https://kafka.apache.org/uses)

- High-throughput messaging.
- Website activity tracking.
- Metric collection.
- Low-latency log aggregation.
- Stream processing (e.g., real-time word count).
- Event sourcing and commit logging.

Topics and Logs

A **topic** is a stream of records. It is stored as partitioned log. The retention period for published records is configurable.

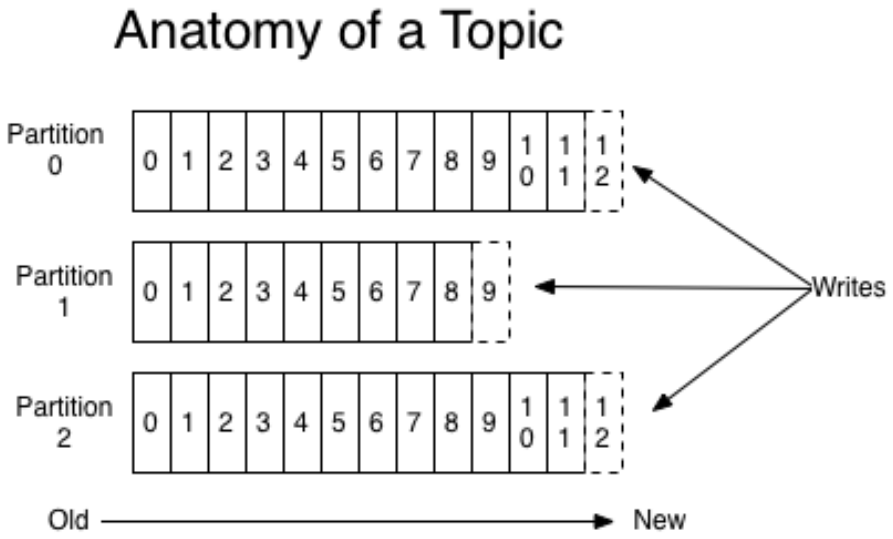


Image source: https://kafka.apache.org/25/images/log_anatomy.png

Topics and Logs

Kafka stores for each consumer the position of the next record to be read. The position is represented as an offset in the log.

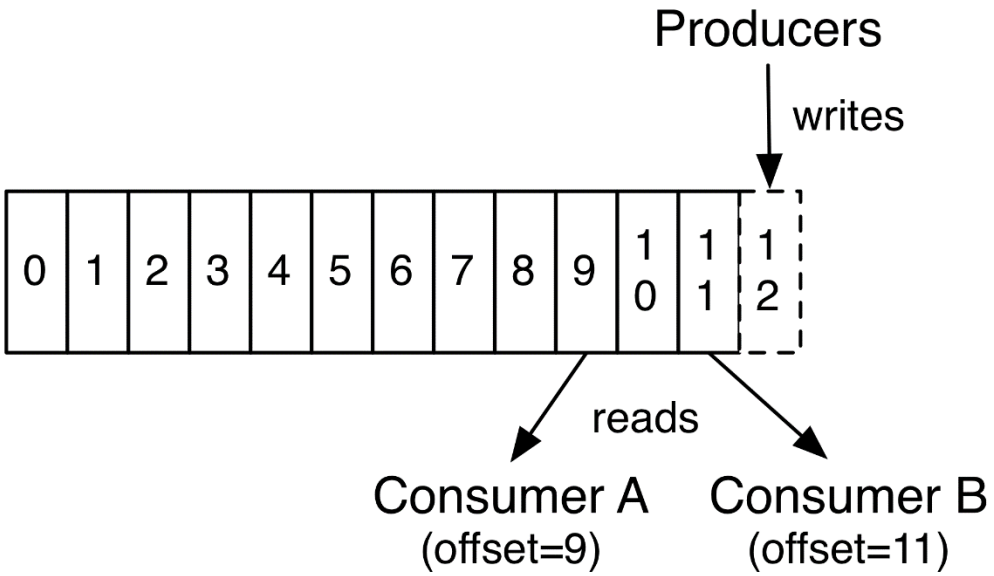


Image source: https://kafka.apache.org/25/images/log_consumer.png

Producers and consumers

Producer:

- pushes records to Kafka brokers, and chooses which partition to contact for a given topic
- can batch records and send them to broker asynchronously (much better throughput with small latency penalty)
- can perform idempotent delivery to avoid duplicate commits

Consumer:

- pulls records in batches from a Kafka broker, who advances the consumer's offset within the topic
- able to achieve "exactly once" semantics when a client consumes from one topic and produces to another

Producer example

Source:
https://kafka.apache.org/25/javadoc/index.html?org/apache/kafka/clients/producer/KafkaProducer.html

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("acks", "all");
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);
for (int i = 0; i < 100; i++)
    producer.send(new ProducerRecord<String, String>
            ("my-topic", Integer.toString(i), Integer.toString(i)));
producer.close();
```

Consumer example (auto-commit)

Source:
https://kafka.apache.org/25/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test");
props.put("enable.auto.commit", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("foo", "bar"));
while (true) {
    ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
    for (ConsumerRecord<String, String> record : records)
   System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(),
record.value());
}
```

Consumer example (manual commit)

Source:
https://kafka.apache.org/25/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html

```
Properties props = new Properties();
    …
    props.put("enable.auto.commit", "false");
    …
    List<ConsumerRecord<String, String>> buffer = new ArrayList<>();
    while (true) {
       ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
       for (ConsumerRecord<String, String> record : records) {
          buffer.add(record);
       }
       if (buffer.size() >= minBatchSize) {
          insertIntoDb(buffer);
          consumer.commitSync();
          buffer.clear();
       }
    }
```

Streams example

Source: https://kafka.apache.org/25/documentation/streams/

```
Properties config = new Properties();
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-application");
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092");
config.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
config.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());

StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> textLines = builder.stream("TextLinesTopic");
KTable<String, Long> wordCounts = textLines
   .flatMapValues(textLine -> Arrays.asList(textLine.toLowerCase().split("\\W+")))
   .groupBy((key, word) -> word)
   .count(Materialized.<String, Long, KeyValueStore<Bytes, byte[]>>as("counts-store"));
wordCounts.toStream().to("WordsWithCountsTopic", Produced.with(Serdes.String(), Serdes.Long()));

KafkaStreams streams = new KafkaStreams(builder.build(), config);
streams.start();
```

Running the word count example

1. Create the input and output topics using the AdminClient Java class or bin/kafka-topics.sh script.
2. Run the WordCountApplication driver program.
3. Run bin/kafka-console-producer.sh
4. Run bin/kafka-console-consumer.sh with the following additional options:

   --property print.key=true
   --property print.value=true
   --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
   --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
5. Type some text into the producer, press enter, and watch the consumer's output.

Record stream vs. changelog stream

Broadly speaking, there are two ways to interpret the semantics of a stream:

1. **Record stream**, where each record represents a state transition (e.g., the balance of account number 123 is increased by $100).
2. **Changelog stream**, where each record represents a state (e.g., account number 123 has balance $100).

These two types of streams have distinct representations in the Kafka Streams API:

1. **KStream** for record streams.
2. **KTable** for changelog streams.

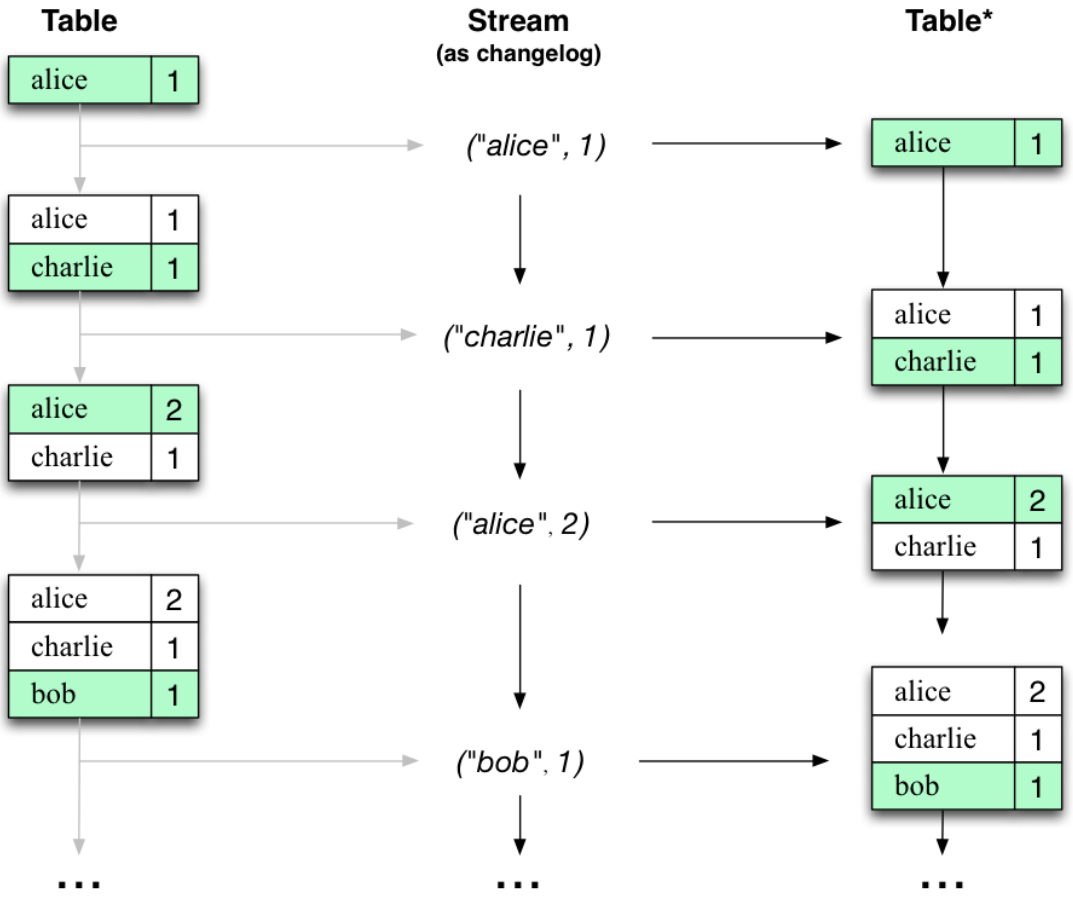
Duality of streams and tables

Changelog streams and tables are interchangeable.

- Each record in a changelog stream defines one row of the table, and overwrites any prior row for the same key.
- A table can be viewed as a snapshot of the latest value for each key in a changelog stream.

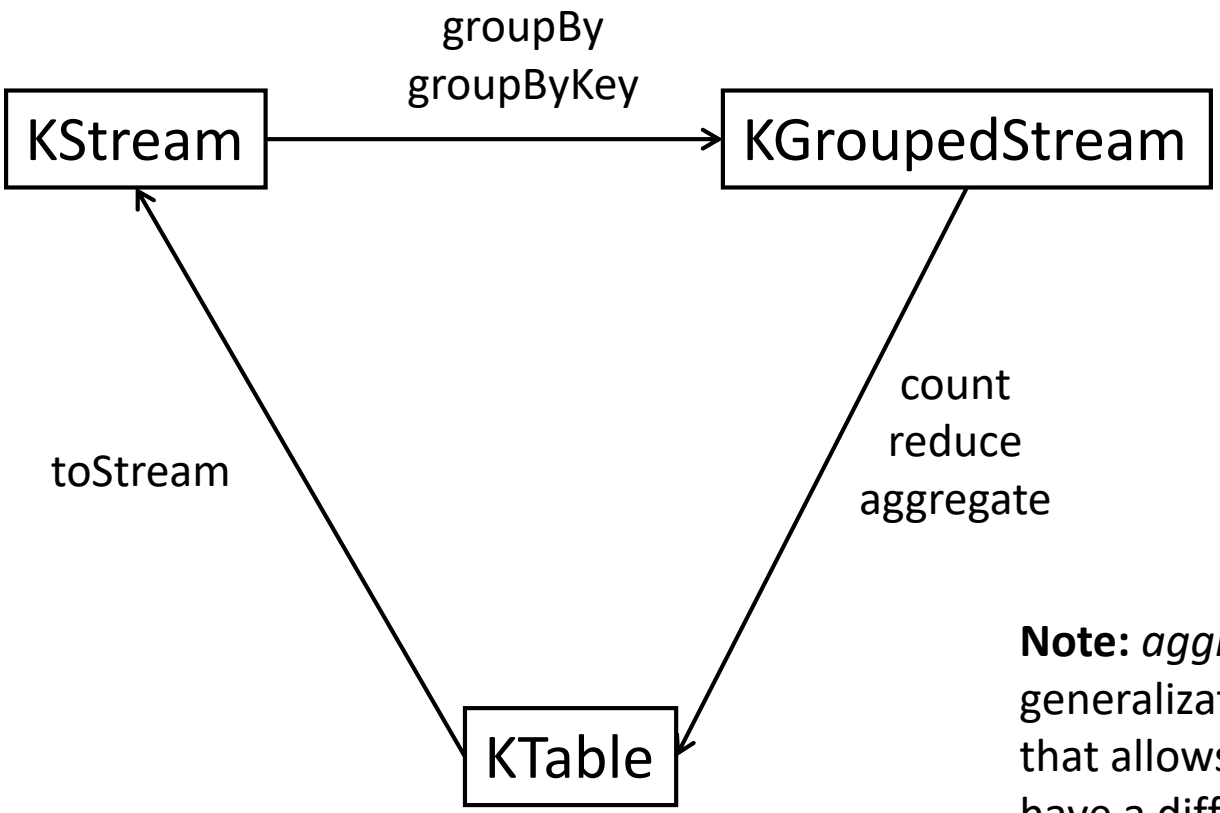
This is why a changelog stream in Kafka is represented by a KTable.

Duality illustration



Source: https://kafka.apache.org/25/images/streams-table-duality-03.png

Converting between KStream and KTable

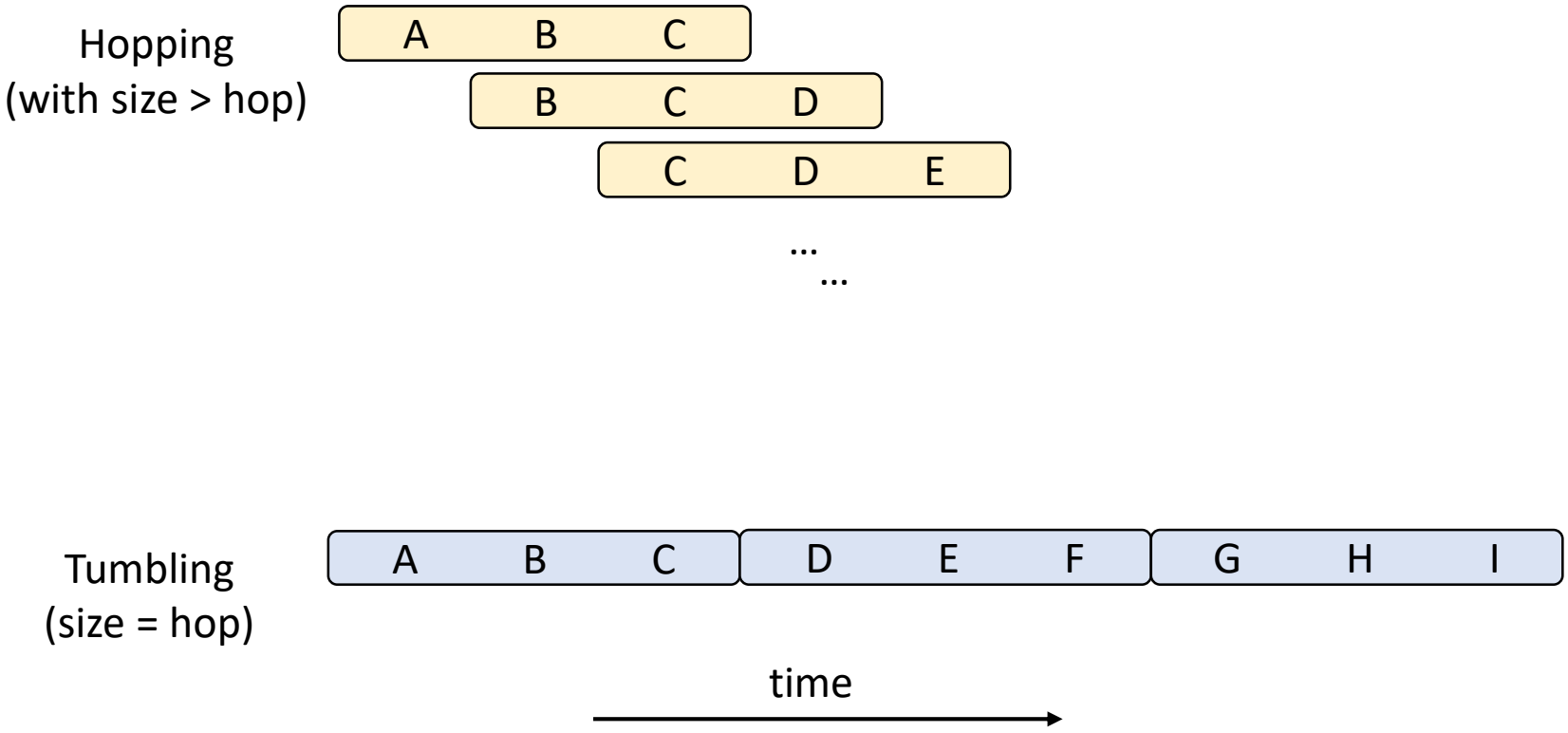


Note: *aggregate* is a generalization of *reduce* that allows the result to have a different type than the input.

Windowed streams

- **Hopping time windows:** defined by size and advance interval ("hop"). For example, every 10 seconds (hop) compute word count over the last 60 seconds of data (size). Hopping windows may be overlapping, or they may have gaps between them.
- **Tumbling time windows:** special case of hopping windows where window size = advance interval. Tumbling windows are non-overlapping and gapless.
- **Sliding windows:** slide continuously over the time axis, used only for joins.
- **Session windows:** aggregate data by period of activity. New session is created when the period of inactivity exceeds a given threshold.

Hopping vs. tumbling windows



Example: tumbling window

```
…
textLines.flatMapValues(textLine -> … )
        .groupBy((key, word) -> word)
         // convert to windowed stream
        .windowedBy(TimeWindows.of(TimeUnit.SECONDS.toMillis(10)))
        .count()
        .toStream()
         // convert windowed key to ordinary key
        .map((key, value) -> KeyValue.pair(key.key(), value))
        .to("WordsWithCountsTopic", Produced.with( … ));
```

Further reading

- https://kafka.apache.org/
- https://www.kafka-online.info/