WIFI

- Jfokus2018 / Jfokus2018
GETTING STARTED

```
git clone https://github.com/fredriv/kafka-streams-workshop.git

cd kafka-streams-workshop

./gradlew build
```
AGENDA

• Intro
• Why streaming?
• Kafka & Kafka Streams
• The basics: Filter, transform and routing
• Next level: Aggregations and windowing
• Joins, enrichments
WHO ARE WE?

- Data Engineers
- Schibsted Data Platform

Fredrik Vraalsen  
Øyvind Løkling
DATA PIPELINE

Collector → Kinesis → Storage → Kafka Streams → S3 → Spark

Collector

Kinesis

Storage

Kafka Streams

S3

Spark
DATA QUALITY

![Data Quality Dashboard]

- **Total Events**
  - Chart showing the total number of events over time.

- **Schema Valid**
  - Chart showing the number of events with valid schemas over time.

**Checks Overview**

**Engage**
- Engage Activity Ping
- Engage Article View
- Engage Visibility Tracking
- Engage Page View

**Predictions**
- User Modeling Optional
- User Modeling

---

*SCHIBSTED*
3RD PARTY ANALYTICS

[Logos of Appboy and AppsFlyer]
DATA-DRIVEN APPLICATIONS

https://www.flickr.com/photos/rahulrodriguez/14683524180


https://www.slideshare.net/DataStax/c-for-deep-learning-andrew-jefferson-tracktable-cassandra-summit-2016
700,000,000 events/day

Lars Marius Garshol, lars.marius.garshol@schibsted.com
2016-09-07, JavaZone 2016
http://twitter.com/larsga
WHY STREAMING?

- Real-time, low latency

https://www.oreilly.com/ideas/questioning-the-lambda-architecture
STREAM PROCESSING

STRAEM PROCESSING

- Unbounded datasets
- Unordered events
- Correctness
- Time
STREAM PROCESSING

Apache Flink

kafka

beam

Spark Streaming

samza

akka

STORM

SCHIBSTED
WHY kafka?
Kafka and the Distributed Log

Anatomy of a Topic

Partition 0
0 1 2 3 4 5 6 7 8 9 1 1 1 0 1 2

Partition 1
0 1 2 3 4 5 6 7 8 9

Partition 2
0 1 2 3 4 5 6 7 8 9 1 1 1 0 1 2

Old ----> New

Writes

http://kafka.apache.org/documentation.html
CONSUMER GROUPS

Partition 1, 2, 3, 4 -> Consumer thread #1
Partition 5, 6, 7, 8 -> Consumer thread #2
Partition 9, 10, 11, 12 -> Consumer thread #3
CONSUMER GROUPS

- Consumer thread #1: Partitions 1, 2, 3, 4
- Consumer thread #2: Partitions 5, 6, 7, 8
- Consumer thread #3: Partitions 9, 10, 11, 12
- Consumer thread #4: Partitions 4, 8, 12
(Key, Value)
LOG EVENTS

(K key, Value)

+ Timestamp, Metadata
KAFKA STREAMS

- Lightweight library
- Streams and tables
- High-level DSL
- Low level API
STREAM PROCESSING COMPONENTS

- Filter
  - Slim down stream, privacy / security
- Transform
  - Bring data into "form", change schema
- Aggregate
  - Compute aggregate values and state (count, sum, etc.)
- Time windows
- Join
  - Enrich by joining datasets
ANATOMY OF A KAFKA STREAMS APP
Properties config = new Properties();
cfg.put(StreamsConfig.APPLICATION_ID_CONFIG, "hello-world-app");
cfg.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");
Properties config = new Properties();
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "hello-world-app");
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");

StreamsBuilder builder = new StreamsBuilder();
// TODO Build topology
ANATOMY OF A KAFKA STREAMS APP

Properties config = new Properties();
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "hello-world-app");
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");

StreamsBuilder builder = new StreamsBuilder();
// TODO Build topology

KafkaStreams streams = new KafkaStreams(topology, config);
streams.start();

Runtime.getRuntime().addShutdownHook(new Thread(() ->
    streams.close(10, TimeUnit.SECONDS)
));
Properties config = new Properties();
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "hello-world-app");
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");

StreamsBuilder builder = new StreamsBuilder();
// TODO Build topology

KafkaStreams streams = new KafkaStreams(topology, config);
streams.start();

Runtime.getRuntime().addShutdownHook(new Thread(() ->
    streams.close(10, TimeUnit.SECONDS)));

SCHIBSTED
HELLO KAFKA STREAMS

StreamBuilder builder = new StreamBuilder();

Serde<String> strings = Serdes.String();
StreamBuilder builder = new StreamBuilder();
Serde<String> strings = Serdes.String();

KStream<String, String> articles = builder.stream("Articles",
    Consumed.with(strings, strings));
HELLO KAFKA STREAMS

StreamBuilder builder = new StreamBuilder();
Serde<String> strings = Serdes.String();

KStream<String, String> articles = builder.stream("Articles",
Consumed.with(strings, strings));

articles.print(Printed.toSysOut());
StreamBuilder builder = new StreamBuilder();

Serde<String> strings = Serdes.String();

KStream<String, String> articles = builder.stream("Articles", Consumed.with(strings, strings));

articles.print(Printed.toSysOut());

2, {"site": "foxnews", "title": "Russia internet wars continue, to Trump's ..."}
3, {"site": "bbc", "title": "Employees urged to let staff 'rest'"}
4, {"site": "cnn", "title": "Italian town sells homes for $1"}
6, {"site": "cnn", "title": "US: More Russia sanctions in 'near future'"}
1, {"site": "cnn", "title": "Ex-Googlers create a self driving car ..."}
5, {"site": "bbc", "title": "What to watch for in Trump's SOTU speech"}
7, {"site": "foxnews", "title": "FBI officials review damning surveillance memo ..."}
8, {"site": "bbc", "title": "The truth about the origin of macaroni cheese"}
StreamBuilder builder = new StreamBuilder();

Serde<String> strings = Serdes.String();
Serde<JsonNode> json = new JsonNodeSerde();

KStream<String, JsonNode> articles = builder.stream("Articles",
Consumed.with(strings, json));

articles.print(Printed.toSysOut());

2, {"site": "foxnews", "title": "Russia internet wars continue, to Trump's ..."}
3, {"site": "bbc", "title": "Employees urged to let staff 'rest'"}
4, {"site": "cnn", "title": "Italian town sells homes for $1"}
6, {"site": "cnn", "title": "US: More Russia sanctions in 'near future'"}
1, {"site": "cnn", "title": "Ex-Googlers create a self driving car ..."}
5, {"site": "bbc", "title": "What to watch for in Trump's SOTU speech"}
7, {"site": "foxnews", "title": "FBI officials review damning surveillance memo ..."}
8, {"site": "bbc", "title": "The truth about the origin of macaroni cheese"}
class JsonNodeSerde extends Serde[JsonNode] {

  override val serializer = new Serializer[JsonNode] {
    override def serialize(topic: String, data: JsonNode): Array[Byte] = ???
    override def configure ...
    override def close ...
  }

  override val deserializer = new Deserializer[JsonNode] {
    override def deserialize(topic: String, data: Array[Byte]): JsonNode = ???
    override def configure ...
    override def close ...
  }

  override def configure ...
  override def close ...
}
class JsonNodeSerde extends Serde[JsonNode] {

  private val mapper = new ObjectMapper() // Jackson JSON API

  override val serializer = new Serializer[JsonNode] {
    override def serialize(topic: String, data: JsonNode): Array[Byte] = mapper.writeValueAsBytes(data)
  }

  override def configure ...
  override def close ...

  override val deserializer = new Deserializer[JsonNode] {
    override def deserialize(topic: String, data: Array[Byte]): JsonNode = mapper.readTree(data)
  }

  override def configure ...
  override def close ...

  override def configure ...
  override def close ...
}
config.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
config.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, JsonNodeSerde.class);

StreamBuilder builder = new StreamBuilder();

KStream<String, JsonNode> articles = builder.stream("Articles");
THE BASICS
KStream<String, JsonNode> articles = builder.stream("Articles",
    Consumed.with(strings, json));
KStream<String, JsonNode> articles = builder.stream("Articles",
        Consumed.with(strings, json));

KStream<String, JsonNode> bbcArticles = articles
    .filter((key, article) -> "bbc".equals(article.path("site").asText()));
KStream<String, JsonNode> articles = builder.stream("Articles", Consumed.with(strings, json));

KStream<String, JsonNode> bbcArticles = articles.filter((key, article) -> "bbc".equals(article.path("site").asText()));

bbcArticles.print(Printed.toSysOut());

3, {"site": "bbc", "title": "Employees urged to let staff 'rest'"}
5, {"site": "bbc", "title": "What to watch for in Trump's SOTU speech"}
8, {"site": "bbc", "title": "The truth about the origin of macaroni cheese"}
KStream<String, JsonNode> articles = builder.stream("Articles", Consumed.with(strings, json));

KStream<String, JsonNode> bbcArticles = articles
    .filter((key, article) -> "bbc".equals(article.path("site").asText()));

KStream<String, String> bbcTitles = bbcArticles
    .mapValues(article -> article.path("title").asText());

bbcTitles.print(Printed.toSysOut());

3, Employees urged to let staff 'rest'
5, What to watch for in Trump's SOTU speech
8, The truth about the origin of macaroni cheese
KStream<String, JsonNode> articles = builder.stream("Articles", Consumed.with(strings, json));

KStream<String, String> bbcTitles = articles.filter((key, article) -> "bbc".equals(article.path("site").asText())).mapValues(article -> article.path("title").asText());

bbcTitles.print(Printed.toSysOut());

3, Employees urged to let staff 'rest'
5, What to watch for in Trump's SOTU speech
8, The truth about the origin of macaroni cheese
KStream<String, JsonNode> articles = builder.stream("Articles", Consumed.with(strings, json));

KStream<String, String> bbcTitles = articles.filter((key, article) -> "bbc".equals(article.path("site").asText())).mapValues(article -> article.path("title").asText());

bbcTitles.to("BBC-Titles", Produced.with(strings, strings));

fredriv@fredriv-mac 20:30:33
$ master $ kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic BBC-Titles --from-beginning
What to watch for in Trump’s SOTU speech
Employees urged to let staff 'rest'
The truth about the origin of macaroni cheese
^CProcessed a total of 3 messages

SCHIBSTED
KStream<String, JsonNode> articles = builder.stream("Articles",
    Consumed.with(strings, json));

KStream<String, String> bbcTitles = articles
    .filter((key, article) -> "bbc".equals(article.path("site").asText()))
    .mapValues(article -> article.path("title").asText());

bbcTitles.to("BBC-Titles", Produced.with(strings, strings));
SCALA GOTCHAS

val articles = builder.stream("Articles",
   Consumed.\`with\`(strings, json))

val bbcTitles = articles
   .filter((key, article) => article.path("site").asText == "bbc")
   .mapValues[String](article => article.path("title").asText)

bbcTitles.to("BBC-Titles", Produced.\`with\`(strings, strings))
import kafkastreams.scalautils.JacksonDSL._
import kafkastreams.scalautils.KafkaStreamsDSL._

val articles = builder.streamS[String, JsonNode]("Articles")

val bbcTitles = articles
  .filter((key, article) => article("site").asText == "bbc")
  .mapValuesS(article => article("title").asText)

bbcTitles.toS("BBC-Titles")
import kafkastreams.scalautils.JacksonDSL._
import kafkastreams.scalautils.KafkaStreamsDSL._

val articles = builder.streamS[String, JsonNode]("Articles")

val bbcTitles = articles \
   ((_("site").asText == "bbc") ~> \
    (_("title").asText))

bbcTitles ~> "BBC-Titles"
TRANSFORM

- Convert 1 input event to sequence of output events
  - Zero, one or many

- flatMap / flatMapValues
  - value -> Iterable
    - e.g. Collection, Arrays.asList, Iterator(ish)
FLATMAP

KStream<String, JsonNode> articles = builder.stream("Articles");

KStream<String, String> authors = articles
    .flatMapValues(json -> json.path("authors").elements())  // fails :-(
    .mapValues(author -> author.asText());
KStream<String, JsonNode> articles = builder.stream("Articles");

KStream<String, String> authors = articles.flatMapValues(json -> new Iterable<JsonNode>() {
    @Override
    public Iterator<JsonNode> iterator() {
        return json.path("authors").elements();
    }
}).mapValues(author -> author.asText());
KStream<String, JsonNode> articles = builder.stream("Articles");

KStream<String, String> authors = articles
    .flatMapValues(json -> () -> json.path("authors").elements())
    .mapValues(author -> author.asText());
KStream<String, JsonNode> articles = builder.stream("Articles");

KStream<String, JsonNode>[] articlesBySite = articles.branch();
BRANCHING

KStream<String, JsonNode> articles = builder.stream("Articles");

KStream<String, JsonNode>[] articlesBySite = articles.branch(
    (key, value) -> "bbc".equals(value.path("site").asText()),
    (key, value) -> "cnn".equals(value.path("site").asText()),
    (key, value) -> "foxnews".equals(value.path("site").asText()),
    (key, value) -> true
);
KStream<String, JsonNode> articles = builder.stream("Articles");

KStream<String, JsonNode>[] articlesBySite = articles.branch(
    (key, value) -> "bbc".equals(value.path("site").asText()),
    (key, value) -> "cnn".equals(value.path("site").asText()),
    (key, value) -> "foxnews".equals(value.path("site").asText()),
    (key, value) -> true
);

articlesBySite[0].to("BBC-Articles");
articlesBySite[1].to("CNN-Articles");
articlesBySite[2].to("FoxNews-Articles");
articlesBySite[3].to("Other-Articles");
EXERCISE 1

• Open Exercise_1_FilterAndTransform
• Implement methods to make tests pass
  • Run in IDE or with: ./gradlew test
  • Tests in Exercise_1_FilterAndTransformTest
  • Test data in ClickEvents

• Should be sufficient with: filter, mapValues, flatMapValues, branch

• For more:
  • https://kafka.apache.org/10/javadoc/index.html
  • https://docs.confluent.io/4.0.0/streams/developer-guide/dsl-api.html
NEXT LEVEL
KStream<String, JsonNode> articles = builder.stream(
"Articles",
Consumed.with(strings, json));

Number of articles per site?
AGGREGATIONS

KStream<String, JsonNode> articles = builder.stream("Articles",
  Consumed.with(strings, json));

KGroupedStream<String, JsonNode> grouped = articles
  .groupBy((key, value) -> value.path("site").asText(),
  Serialized.with(strings, json));
KStream<String, JsonNode> articles = builder.stream("Articles",
    Consumed.with(strings, json));

KGroupedStream<String, JsonNode> grouped = articles
    .groupBy((key, value) -> value.path("site").asText(),
        Serialized.with(strings, json));

KTable<String, Long> counts = grouped.count();
KStream<String, JsonNode> articles = builder.stream("Articles", Consumed.with(strings, json));

KTable<String, Long> articlesPerSite = articles
    .groupBy((key, value) -> value.path("site").asText(),
             Serialized.with(strings, json))
    .count();

articlesBySite.toStream().print(Printed.toSysOut());

foxnews, 2
cnn, 3
bbc, 3
TABLES VS STREAMS

http://kafka.apache.org/documentation/streams/developer-guide#streams_duality
TABLES VS STREAMS

http://kafka.apache.org/documentation/streams/developer-guide#streams_duality
REPARTITIONING

Narrow transformation
- Input and output stays in same partition
- No data movement is needed

Wide transformation
- Input from other partitions are required
- Data shuffling is needed before processing

REPARTITIONING

- map, flatMap, selectKey
- groupByKey
- groupBy
WINDOWED AGGREGATIONS

A 5-min Tumbling Window

Data records

Same colour means same record key

Windows are created per record key

https://docs.confluent.io/current/streams/developer-guide.html#windowing
```java
KStream<String, JsonNode> articles = builder.stream("Articles", Consumed.with(strings, json));

KGroupedStream<String, JsonNode> grouped = articles
    .groupBy((key, value) -> value.path("site").asText(),
                Serialized.with(strings, json));

KTable<Windowed<String>, Long> articlesPerHour = grouped
    .windowedBy(TimeWindows.of(TimeUnit.HOURS.toMillis(1)))
    .count(Materialized.as("articles-per-hour"));
```
EXERCISE 2

• Open Exercise_2_Aggregations

• Implement methods to make tests pass
  • Run in IDE or with: ./gradlew test
  • Tests in Exercise_2_AggregationTest
  • Test data in ClickEvents

• Should be sufficient with:
  • map, selectKey, groupBy, groupByKey, count, reduce
CONNECTING TO THE WORLD
QUERYABLE STATE STORES

1. Capture business events in Kafka
2. Process the events with Kafka Streams
3. With interactive queries, other apps can directly query the latest results

QUERYING STATE STORES

ReadOnlyWindowStore<String, Long> articlesPerHour =
    streams.store("articles-per-hour", QueryableStoreTypes.windowStore());
ReadOnlyWindowStore<String, Long> articlesPerHour = 
    streams.store("articles-per-hour", QueryableStoreTypes.windowStore());

long from = 0L; // Jan 1st 1970
long to = System.currentTimeMillis();
WindowStoreIterator<Long> articles = articlesPerHour.fetch("bbc", from, to);
QUERYING STATE STORES

```java
ReadOnlyWindowStore<String, Long> articlesPerHour = 
    streams.store("articles-per-hour", QueryableStoreTypes.windowStore());

long from = 0L;  // Jan 1st 1970
long to = System.currentTimeMillis();
WindowStoreIterator<Long> articles = articlesPerHour.fetch("bbc", from, to);

articles.forEachRemaining(kv -> {
    Instant timestamp = Instant.ofEpochMilli(kv.key);
    System.out.println("BBC published " + kv.value +
                        " article(s) in hour " + timestamp);
});

BBC published 1 article(s) in hour 2018-02-04T22:00:00Z
```
QUERYABLE STATE STORES

GETTING DATA IN AND OUT

GETTING DATA IN AND OUT

http://kafka.apache.org/documentation.html
JOINS AND ENRICHMENTS
JOINING STREAMS

https://churchtecharts.org/home/2014/10/5/important-safety-tip-with-dante-dont-cross-the-streams
# JOINS

<table>
<thead>
<tr>
<th>Join operands</th>
<th>Type</th>
<th>(INNER) JOIN</th>
<th>LEFT JOIN</th>
<th>OUTER JOIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>KStream-to-KStream</td>
<td>Windowed</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>KTable-to-KTable</td>
<td>Non-windowed</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>KStream-to-KTable</td>
<td>Non-windowed</td>
<td>Supported</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
<tr>
<td>KStream-to-GlobalKTable</td>
<td>Non-windowed</td>
<td>Supported</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
<tr>
<td>KTable-to-GlobalKTable</td>
<td>N/A</td>
<td>Not Supported</td>
<td>Not Supported</td>
<td>Not Supported</td>
</tr>
</tbody>
</table>
KStream<String, String> reads = builder.stream("ArticleReads", Consumed.with(strings, strings));

KTable<String, JsonNode> users = builder.table("Users", Consumed.with(strings, json));
JOINS

KStream<String, String> reads = builder.stream("ArticleReads", Consumed.with(strings, strings));

KTable<String, JsonNode> users = builder.table("Users", Consumed.with(strings, json));

KStream<String, JsonNode> readsByCountry = reads.join(users, (article, user) -> user.path("country"));
JOINS

KStream<String, String> reads = builder.stream("ArticleReads",
    Consumed.with(strings, strings));

KTable<String, JsonNode> users = builder.table("Users",
    Consumed.with(strings, json));

KStream<String, JsonNode> readsByCountry = reads
    .join(users, (article, user) -> user.path("country"));

KTable<String, Long> readsPerCountry = readsByCountry
    .groupBy((userId, country) -> country.asText(),
        Serialized.with(strings, json))
    .count();
JOINS

https://www.confluent.io/blog/crossing-streams-joins-apache-kafka/
ENRICHMENTS

• Enrich events with information from external sources
• Call 3rd party service in mapValues
• Use Processor API for batching
TRANSLATING 700 MILLION EVENTS FROM IP TO COORDINATES EACH DAY USING KAFKA STREAMS

JavaZone 2017
Håkon Åmdal, 14.09.2017
QUESTIONS?
FURTHER READING

• https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
• https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102
• https://www.confluent.io/blog/
• https://www.confluent.io/blog/crossing-streams-joins-apache-kafka/
• https://docs.confluent.io/current/streams/
• http://kafka.apache.org/documentationStreams/