可用资源

Kafka Connect Introduction

© Stephane Maarek

- Do you feel you're not the first person in the world to write a way to get data out of Twitter?
- Do you feel like you're not the first person in the world to send data from Kafka to PostgreSQL / ElasticSearch / MongoDB ?
- Additionally, the bugs you'll have, won't someone have fixed them already?
- Kafka Connect is all about code & connectors re-use!

可用资源

Kafka Connect API
A brief history

© Stephane Maarek

- (2013) Kafka 0.8.x:
  - Topic replication, Log compaction
  - Simplified producer client API
- (Nov 2015) Kafka 0.9.x:
  - Simplified high level consumer APIs, without Zookeeper dependency
  - Added security (Encryption and Authentication)
  - **Kafka Connect APIs**
- (May 2016): Kafka 0.10.0:
  - **Kafka Streams APIs**
- (end 2016 – March 2017) Kafka 0.10.1, 0.10.2:
  - **Improved Connect API, Single Message Transforms API**

可用资源

Why Kafka Connect and Streams

© Stephane Maarek

- Four Common Kafka Use Cases:

Source =>	Kafka	Producer API	**Kafka Connect Source**
Kafka =>	Kafka	Consumer, Producer API	Kafka Streams
Kafka =>	Sink	Consumer API	**Kafka Connect Sink**
Kafka =>	App	Consumer API	

- Simplify and improve getting data in and out of Kafka
- Simplify transforming data within Kafka without relying on external libs

可用资源

Why Kafka Connect

© Stephane Maarek

- Programmers always want to import data from the same sources:

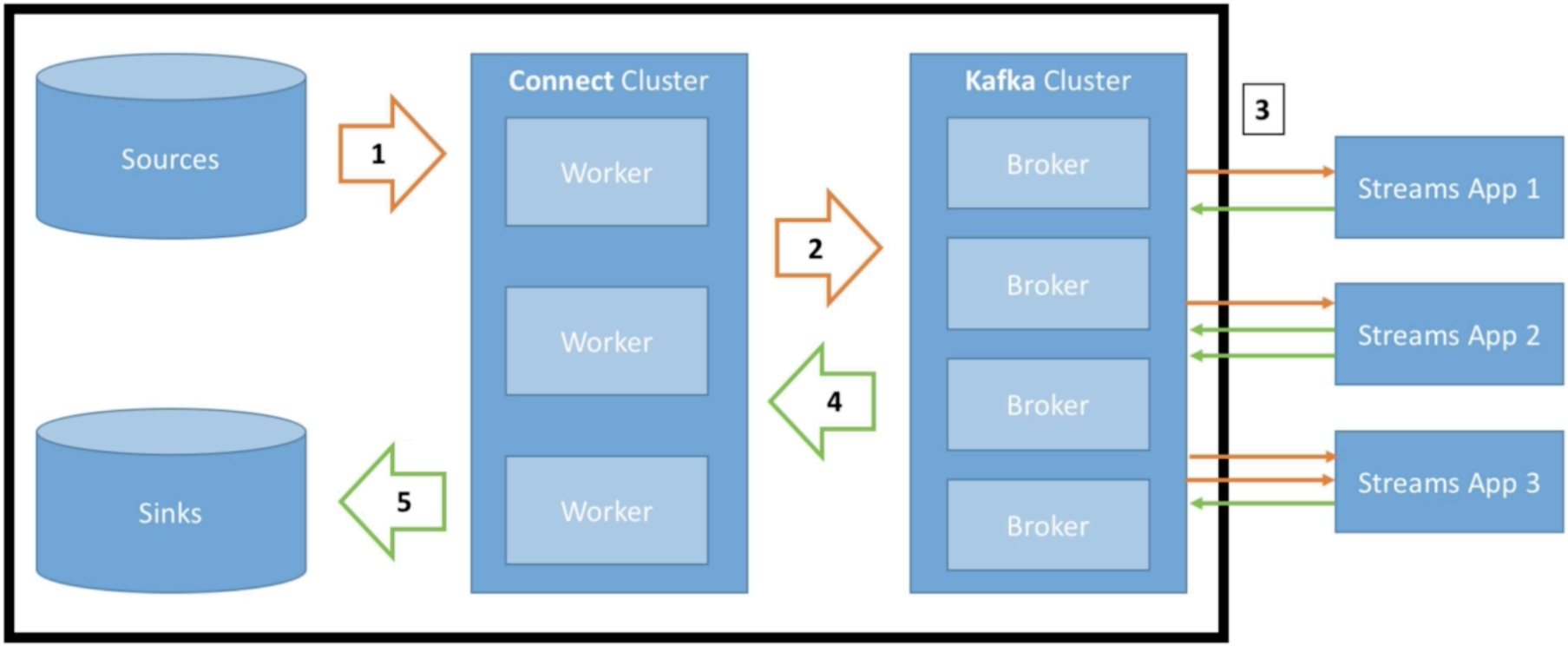
  Databases, JDBC, Couchbase, GoldenGate, SAP HANA, Blockchain, Cassandra, DynamoDB, FTP, IOT, MongoDB, MQTT, RethinkDB, Salesforce, Solr, SQS, Twitter, etc…

- Programmers always want to store data in the same sinks:
  - S3, ElasticSearch, HDFS, JDBC, SAP HANA, DocumentDB, Cassandra, DynamoDB, HBase, MongoDB, Redis, Solr, Splunk, Twitter
- It is tough to achieve Fault Tolerance, Idempotence, Distribution, Ordering
- Other programmers may already have done a very good job!

可用资源

Kafka Connect and Streams Architecture Design

© Stephane Maarek



可用资源

Kafka Connect – High level

© Stephane Maarek

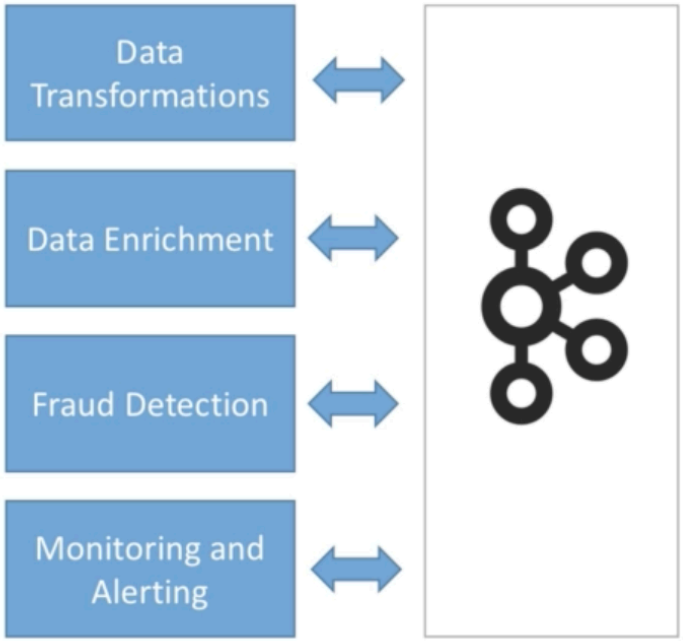
- Source Connectors to get data from Common Data Sources
- Sink Connectors to publish that data in Common Data Stores
- Make it easy for non-experienced dev to quickly get their data reliably into Kafka
- Part of your ETL pipeline
- Scaling made easy from small pipelines to company-wide pipelines
- Re-usable code!

Kafka Streams Introduction

- You want to do the following from the twitter_tweets topic:
  - Filter only tweets that have over 10 likes or replies
  - Count the number of tweets received for each hashtag every1 minute
- Or combine the two to get trending topics and hashtags in real time!
- With Kafka Producer and Consumer, you can achieve that but it's very low level and not developer friendly

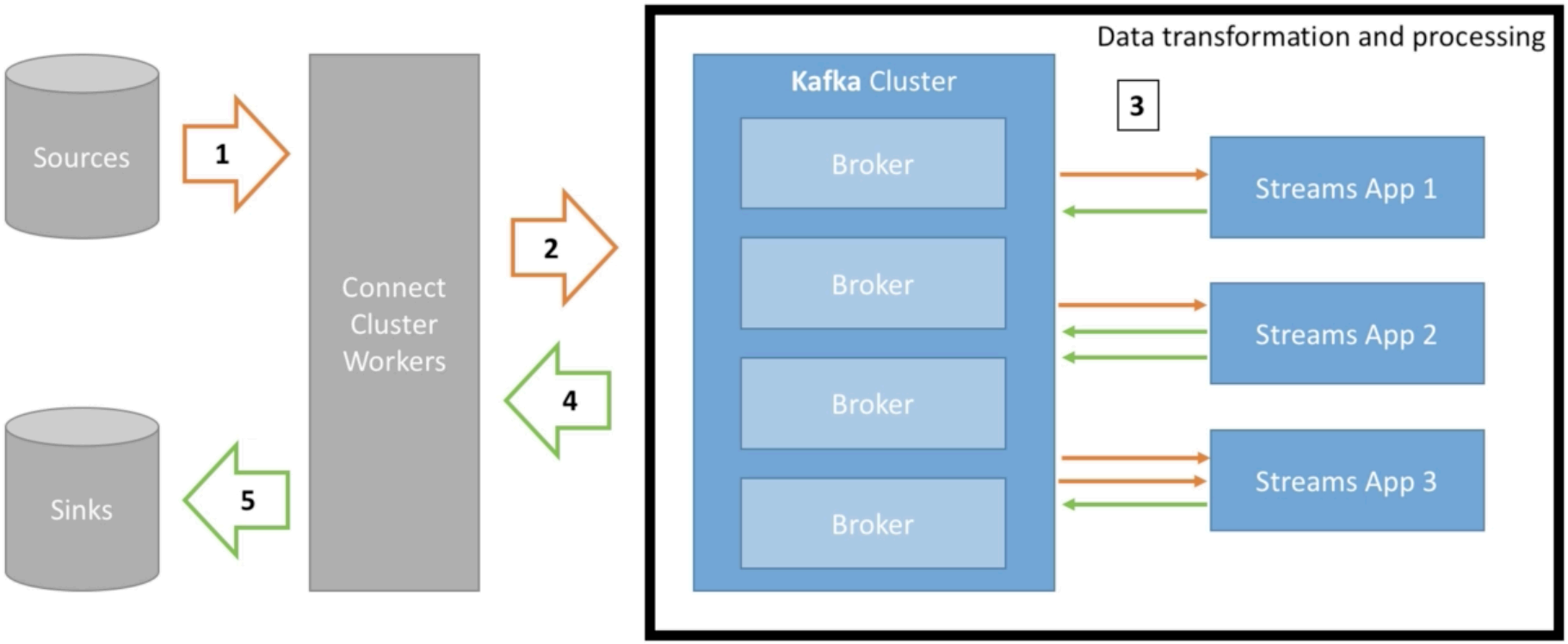
What is Kafka Streams ?

- Easy **data processing and transformation library** within Kafka



- Standard Java Application
- No need to create a separate cluster
- Highly scalable, elastic and fault tolerant
- Exactly Once Capabilities
- One record at a time processing (no batching)
- Works for any application size

Kafka Streams Architecture Design



© Stephane Maarek

Kafka Streams history

© Stephane Maarek

- This API / Library was introduced as part of Kafka 0.10 (XX 2016) and is fully mature as part of Kafka 0.11.0.0 (June 2017)
- It's the only library at this time of writing that can leverage the new exactly once capabilities from Kafka 0.11
- It is a serious contender to other processing frameworks such as Apache Spark, Flink, or NiFi
- Active library so prone to some changes in the future

Example Tweets Filtering

© Stephane Maarek

- We want to filter a tweets topic and put the results back to Kafka
- We basically want to chain a Consumer with a Producer

- This is complicated and error prone, especially if you want to deal with concurrency and error scenarios

The need for a schema registry

© Stephane Maarek

- What if the producer sends bad data?
- What if a field gets renamed?
- What if the data format changes from one day to another?

- **The Consumers Break!!!**

The need for a schema registry

© Stephane Maarek

- We need data to be self describable
- We need to be able to evolve data without breaking downstream consumers.

- We need schemas… and a schema registry!

The need for a schema registry

© Stephane Maarek

- What if the Kafka Brokers were verifying the messages they receive?

- It would break what makes Kafka so good:
  - Kafka doesn't parse or even read your data (no CPU usage)
  - Kafka takes bytes as an input without even loading them into memory (that's called zero copy)
  - Kafka distributes bytes
  - As far as Kafka is concerned, it doesn't even know if your data is an integer, a string etc.

The need for a schema registry

© Stephane Maarek

- The Schema Registry has to be a separate components
- Producers and Consumers need to be able to talk to it
- The Schema Registry must be able to reject bad data
- A common data format must be agreed upon
  - It needs to support schemas
  - It needs to support evolution
  - It needs to be lightweight
- Enter… the Confluent Schema Registry
- And Apache Avro as the data format.

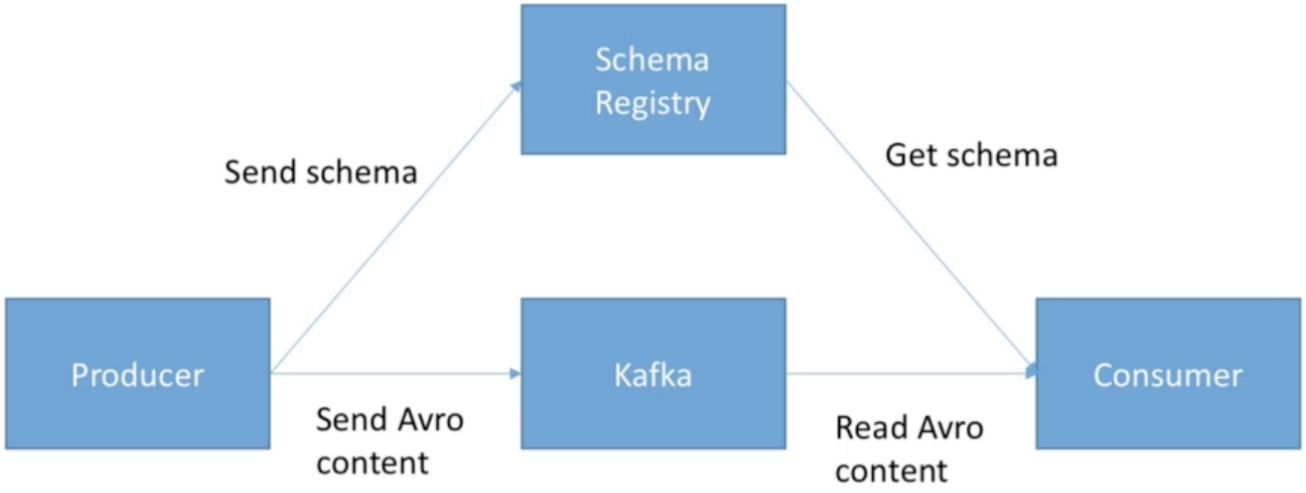
Pipeline without Schema Registry

© Stephane Maarek

Confluent Schema Registry Purpose

© Stephane Maarek

- Store and retrieve schemas for Producers / Consumers
- Enforce Backward / Forward / Full compatibility on topics
- Decrease the size of the payload of data sent to Kafka



Schema Registry: gotchas

© Stephane Maarek

- Utilizing a schema registry has a lot of benefits
- BUT it implies you need to
  - Set it up well
  - Make sure it's highly available
  - Partially change the producer and consumer code
- Apache Avro as a format is awesome but has a learning curve
- The schema registry is free and open sourced, created by Confluent (creators of Kafka)
- As it takes time to setup, we won't cover the usage in this course