

---

# KAFKA AND SPARK

---

Adam Bellemare

---

---

# ABOUT ME

---

University of Waterloo 2009

Senior Software Developer

I work at



---

Part 1: Kafka Service Overview

Part 2: Spark and Kafka in Action at *flipp*


---

---

# WHAT IS KAFKA?

---

Distributed Publish-Subscribe system

Originally developed by **LinkedIn** 

High availability

High data durability

---

# TECHNICALS

---

Extremely high throughput

Very low latency

Multiple language clients

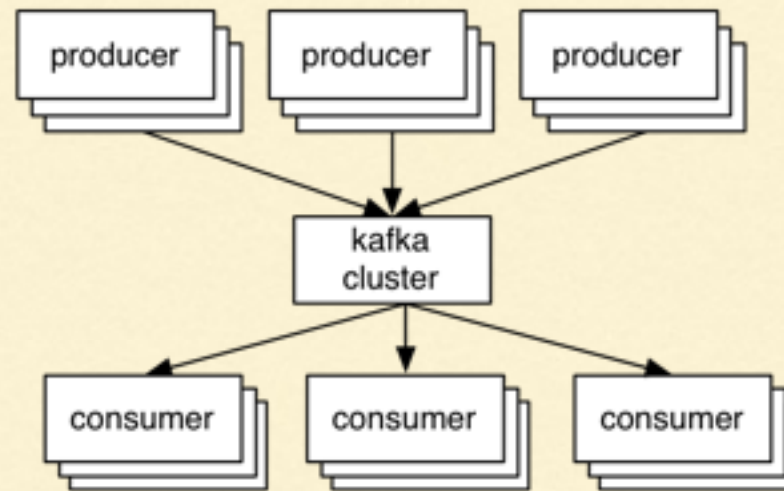
---

Throughput: 1,000,000+ Records / Second

Latency: 99% <= 3 mS, 99.9%: <=14mS

Python, C/C++, Ruby, Node.js, Scala, Java, PHP, stdout/stdin, HTTP/REST, etc.

# OVERVIEW



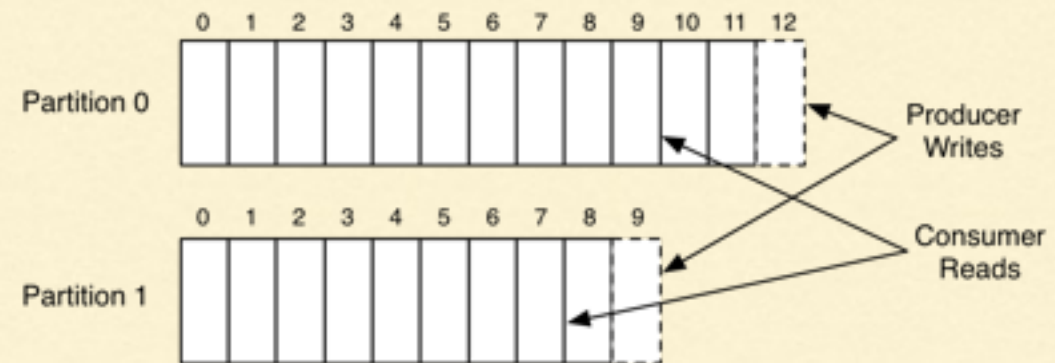
All parts are interdependent, hard to talk about one in isolation without referencing another.

I will cover them all in order as best I can, but since I could start anywhere in the system I will need to hold off on certain details while I describe other parts.

# TOPICS

## Fundamental Division of Data

### Key-Value pair structure



---

# PRODUCERS

---

Produces data to a topic

Partitions the data

ie: User-ID to organize locality

Speed vs. Integrity

- No confirmation
- Confirm once
- Confirm all replications

Plain-text and Avro serialization available

---



---

# BROKERS

---

Kafka cluster is composed of Brokers

Data R/W for each Topic

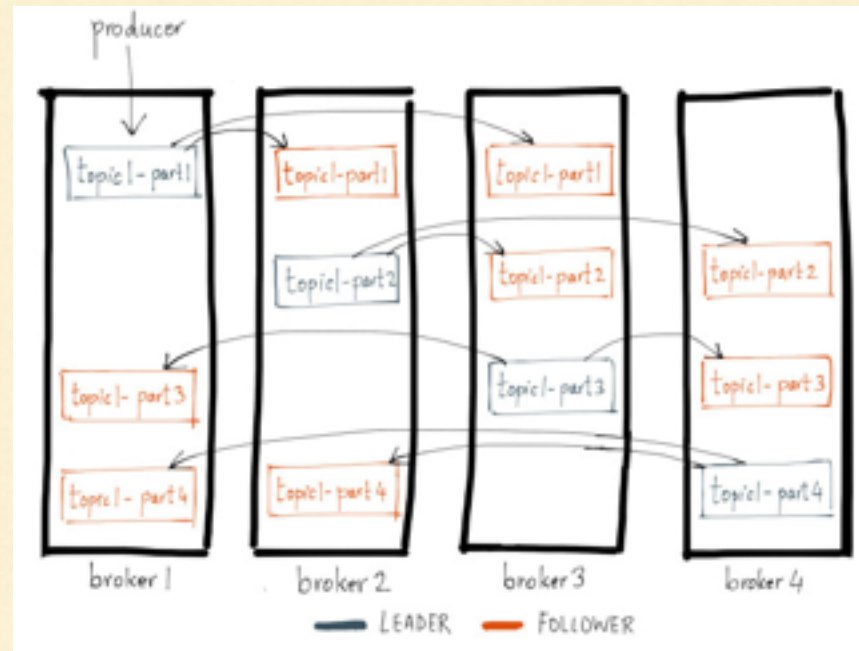
Manages Production

Manages Consumption

Manages Storage

---

# BROKER-TOPIC REPLICATION



Broker Failure

Requires replication (tolerate  $\#Replications-1$  failures)

Topic Leader-election

Replication bootstrapping (restore Topic replication factors)

---

# CONSUMERS

---

Consumes topic data

Can belong to a Consumer-Group

- Uses common topic index
- Each consumer can read a partition

Can re-read previously read data

---

# CONSUMER CONFIGURATIONS

Possible Message Delivery Guarantees:

At-Most-Once

At-Least-Once

Exactly-Once

IDEMPOTENT.

At-Most-Once -> Probably not that useful, since a message you don't receive isn't that valuable.

At-Least-Once -> Idempotent.

Exactly-Once -> Relies on atomic storage of offset by client, plus some additional manual client steps.

# HIGH AVAILABILITY

---

Zookeeper

Topic Leaders Election

Topic Configurations

Replication

---

Kafka uses Zookeeper for the following:

Electing a controller. The controller is one of the brokers and is responsible for maintaining the leader/follower relationship for all the partitions. When a node shuts down, it is the controller that tells other replicas to become partition leaders to replace the partition leaders on the node that is going away.

Zookeeper is used to elect a controller, make sure there is only one and elect a new one if it crashes.

Cluster membership - which brokers are alive and part of the cluster? this is also managed through ZooKeeper.

Topic configuration - which topics exist, how many partitions each has, where are the replicas, who is the preferred leader, what configuration overrides are set for each topic

(0.9.0) - Quotas - how much data is each client allowed to read and write

(0.9.0) - ACLs - who is allowed to read and write to which topic

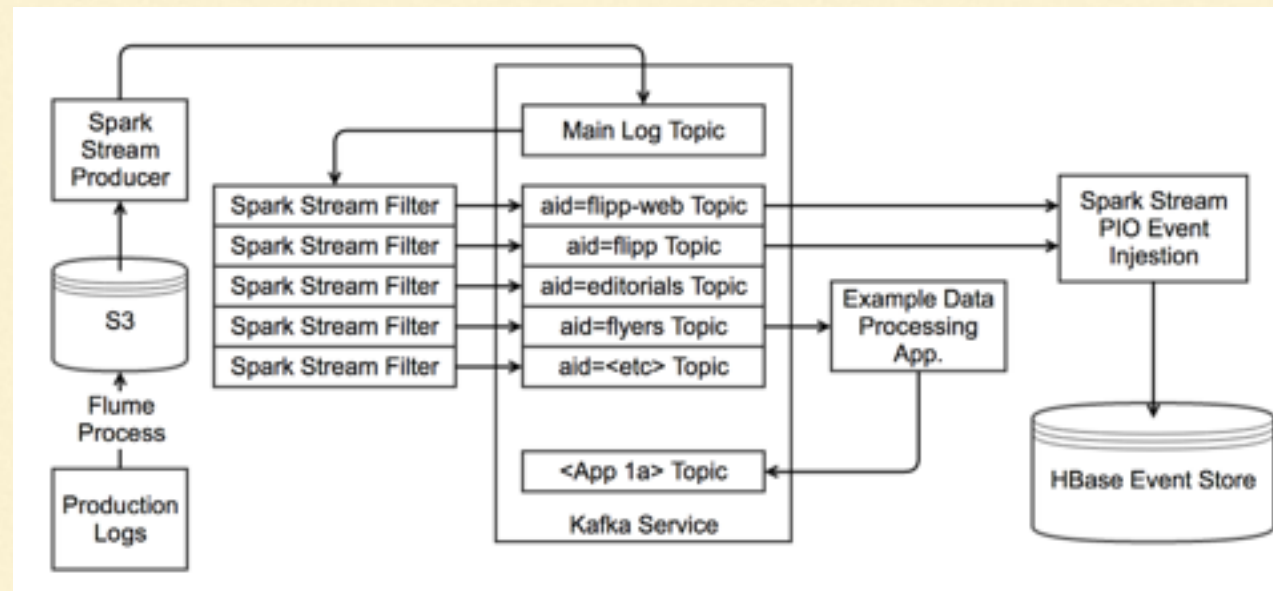
(old high level consumer) - Which consumer groups exist, who are their members and what is the latest offset each group got from each partition.

---

Kafka at *flipp*

---

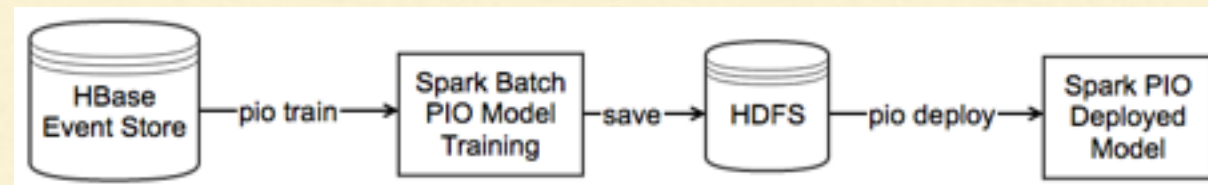
# FLIPP SPARK-KAFKA ARCHITECTURE



# PREDICTION IO WORKFLOW

Prediction IO - Spark-based machine learning framework

Our Models don't directly use Kafka... yet.



Trains models based on events stored in HBase.

Idempotent event model.

Serves recommendations via HTTP



# FLUME TO S3 VS. FLAFKA

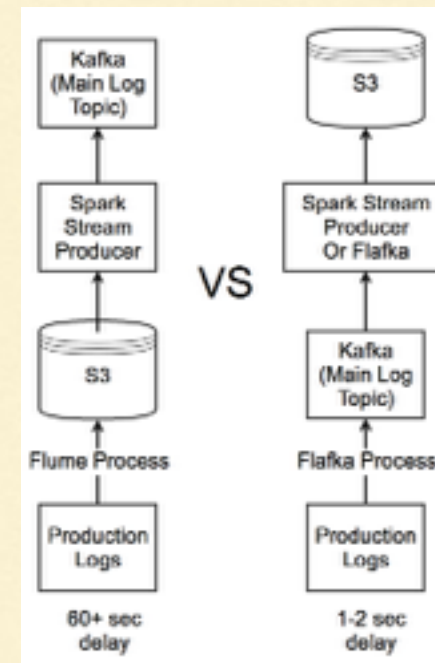
Flakfa = Flume + Kafka

Did not implement it.

System Stability Risks

Billing Risks

Technical Limitations



Stability Risks: Kafka is new to us and not something we want to put in a critical portion of our system without understanding the full implications.

Billing Risks: Changing production systems. Logging data is used to compute billing. Risk of interrupting data flow.

Technical Limitations: You can add interceptors to do processing of data in flight, but there were concerns about how scalable this would be for very large logs.

# AVRO FORMAT

Key-Value

JSON and Binary formats

Clients must encode / decode

Plain Text:

2016-04-27T19:00:00-05:00^Buserid=AAFF^Bdata\_array^Cevent=click^Bsubevent=3^C

Avro JSON Formatted:

{"date": "2016-04-27", "time": "19:00:00-5:00", "userid": "AAFF", "event": "click", "subevent": 3}

Encourage the same data-model usage across the company.

Eliminates the need for duplicate parsing (both Language and Application Level).

Can manage Avro Schemas authoritatively.

# MONITORING KAFKA

---

Burrow (Developed by **LinkedIn**)

Sliding windows, not thresholds

Alerts

Kafka-Committed Offsets (0.82 + )

---

Monitors the Kafka Topic.

Watches incoming vs outgoing messages.

Can notify when a queue starts to grow faster than it is processed, or if processing appears to be stalled (ie: locked up consumer)

---

# PITFALLS

---

Logging Issues -> Long-running jobs having their logs cleaned up due to cron jobs.

Naming management:

Kafka Consumer Group naming -> Create a standard.

Topic Name Management -> Create a standard.

---

We had two processes consuming on the same default group name without either one knowing about the other. Result: Each one gets half the data.

---

# Questions?

Check us out in the app stores (iOS and Android)

Yes, we are hiring!



Check out [corp.flipp.com/careers](https://corp.flipp.com/careers)

---