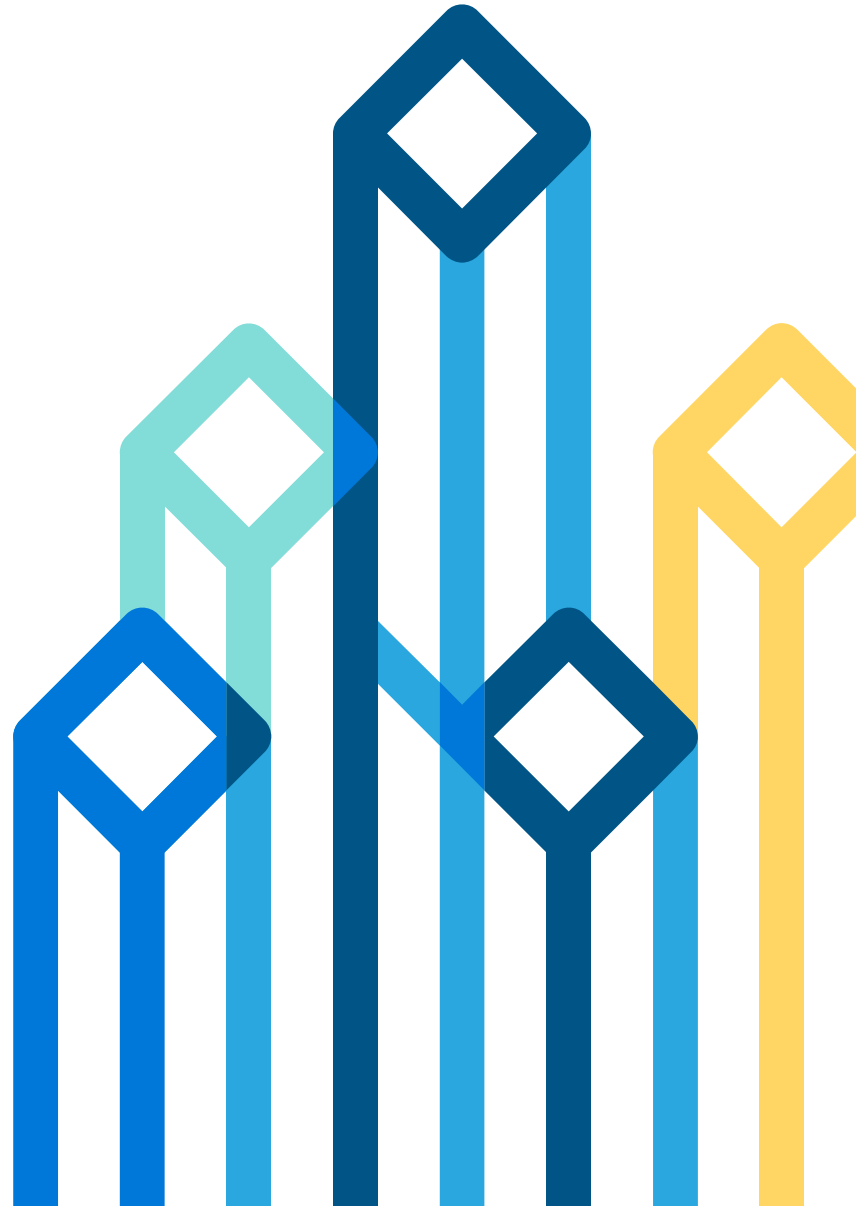
cloudera

Kafka Introduction

Apache Kafka ATL Meetup

Jeff Holoman



Agenda

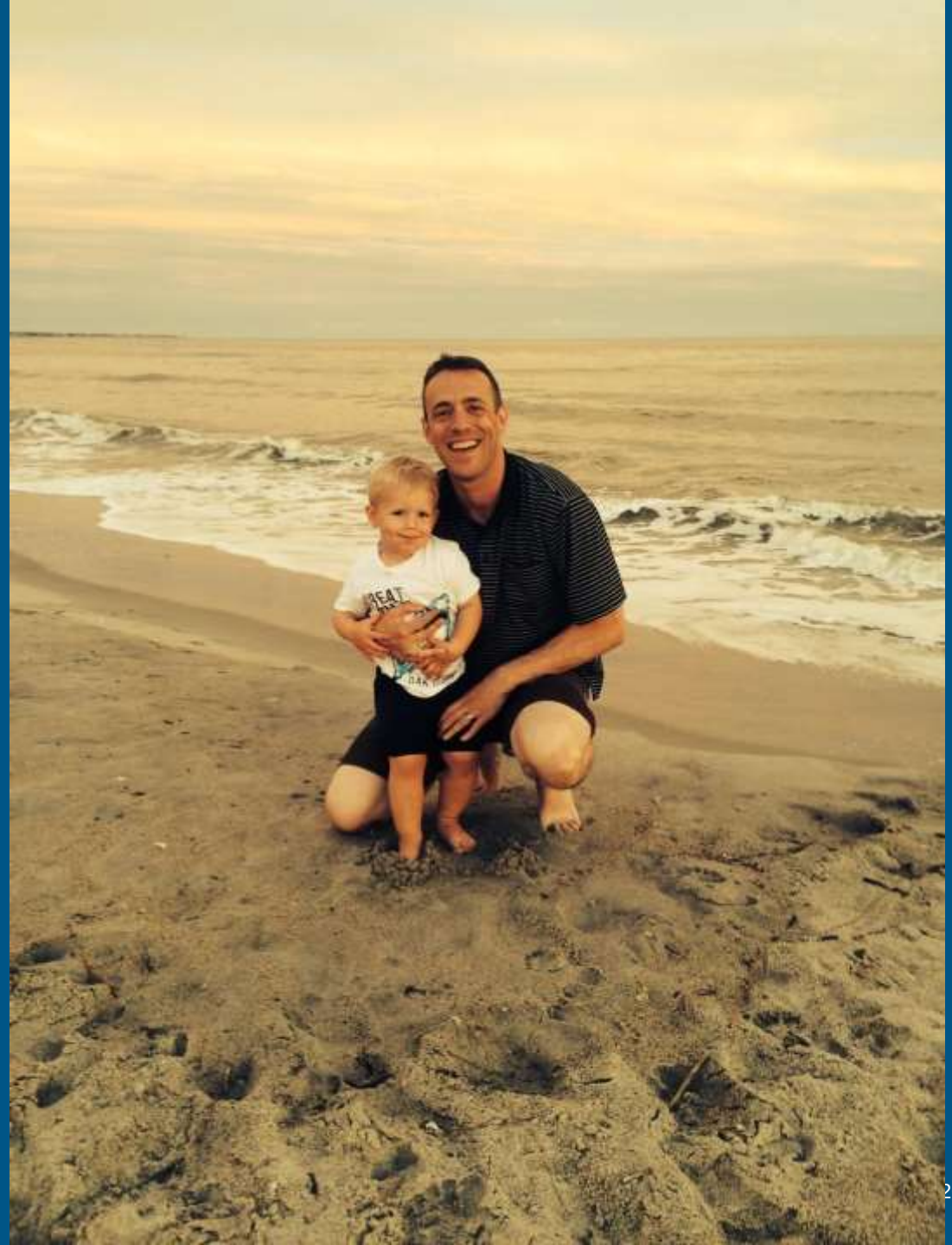
- Introduction
- Concepts
- Use Cases
- Development
- Administration
- Cluster Planning
- Cloudera Integration and Roadmap (New Stuff)

Obligatory "About Me" Slide

Sr. Systems Engineer
Cloudera
@jeffholoman

Areas of Focus
- Data Ingest
- Financial Services
- Healthcare

I mostly spend my time now helping build large-scale solutions in the Financial Services Vertical



Apache Kafka Contributor
Co-Namer of "Flafka"
Cloudera engineering blogger

Former Oracle
- Developer
- DBA

cloudera

Before we get started

- Kafka is picking up steam in the community, a ***lot*** of things are changing. (patches welcome!)
- dev@kafka.apache.org
- users@kafka.apache.org
- http://ingest.tips/ - Great Blog
- Things we won't cover in much detail
  - Wire protocol
  - Security–related stuff (KAFKA-1682)
  - Mirror-maker
- Lets keep this interactive.
- If you are a Kafka user and want to talk at the next one of these, please let me know.

Concepts

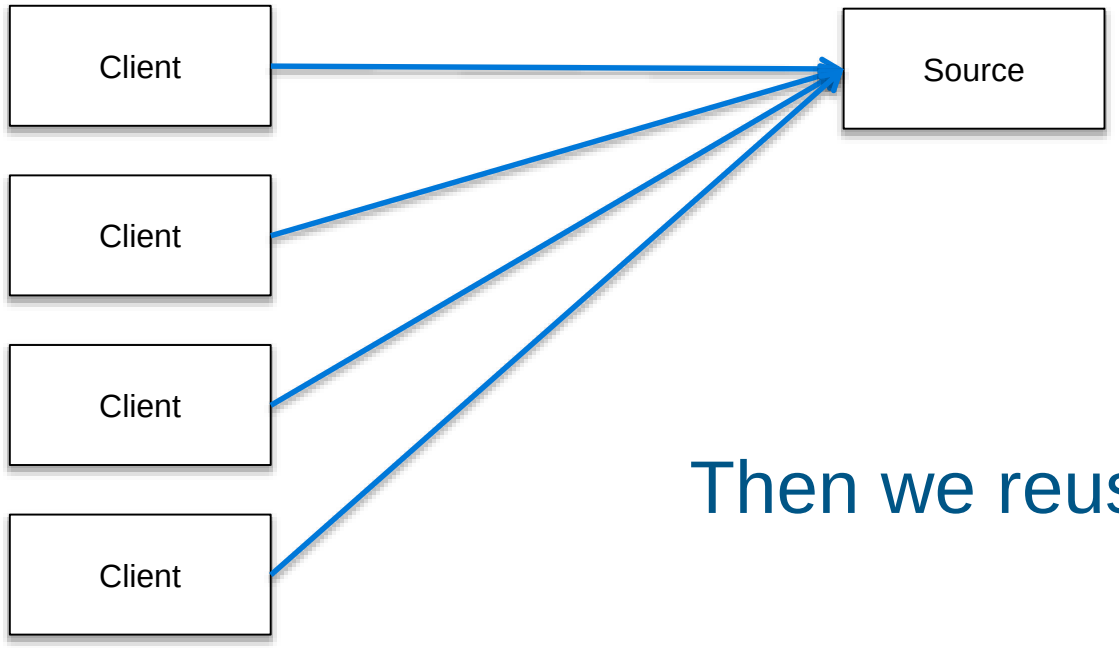
Basic Kafka Concepts

cloudera

Why Kafka

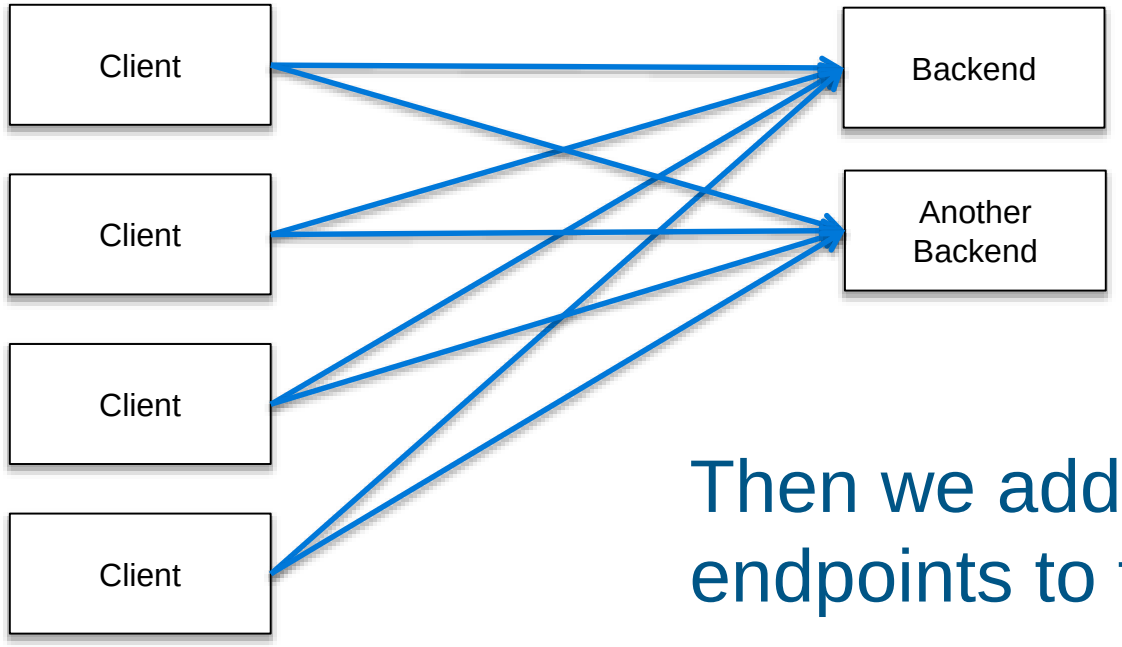
Data Pipelines Start like this.

Why Kafka



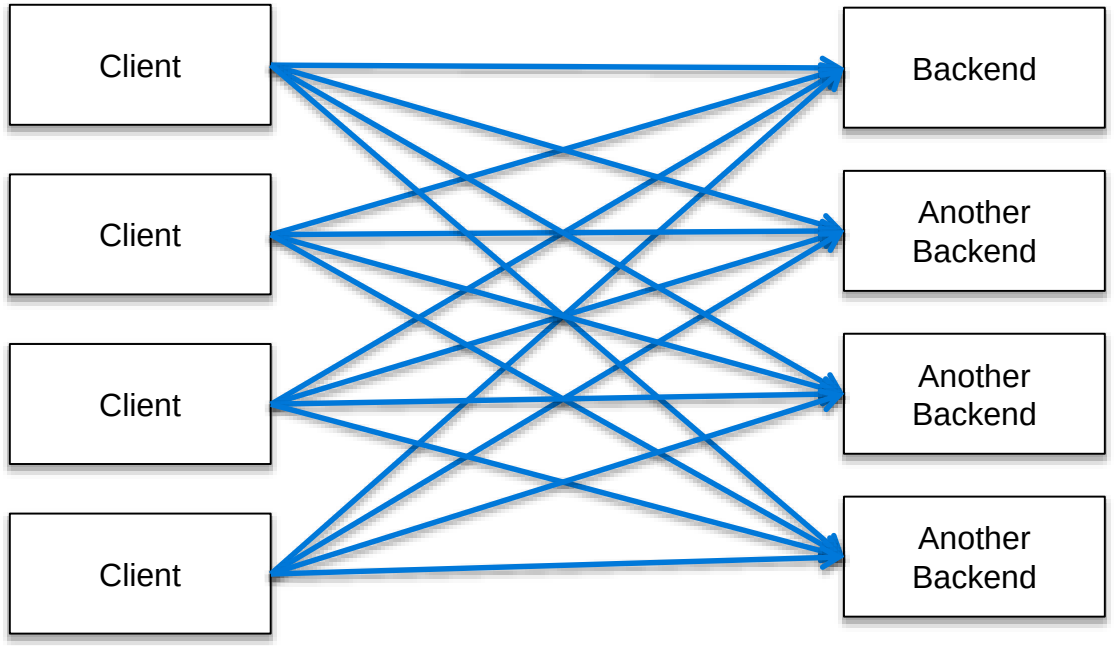
Then we reuse them

Why Kafka



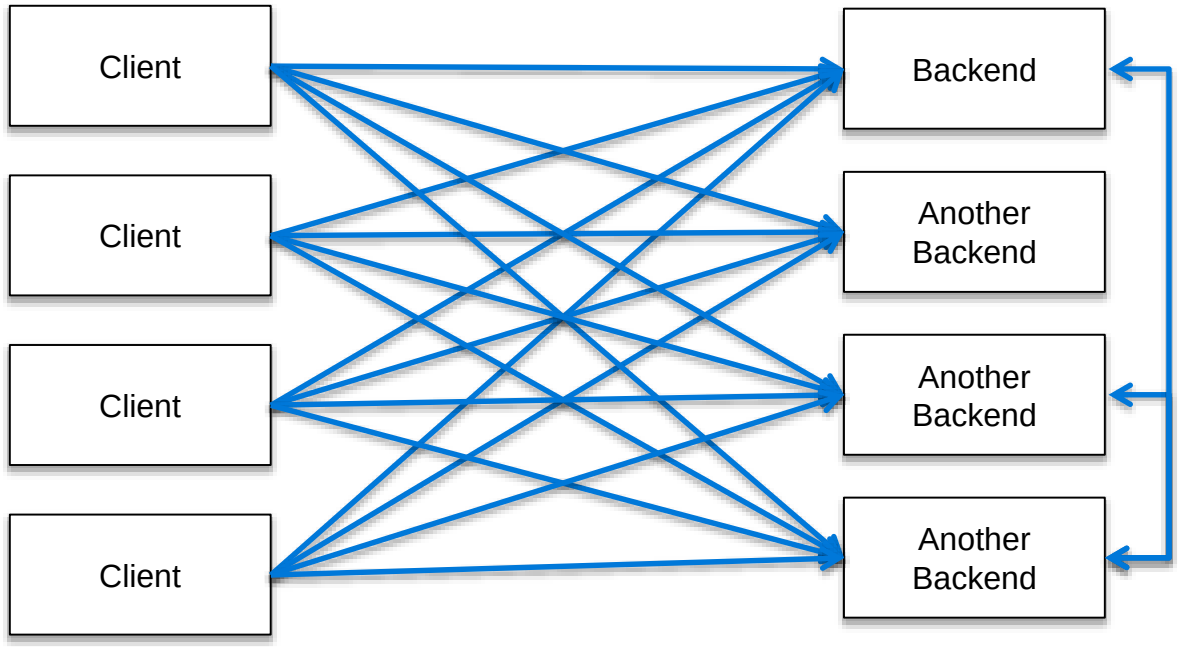
Then we add additional endpoints to the existing sources

Why Kafka



Then it starts to look like this

Why Kafka



With maybe some of this

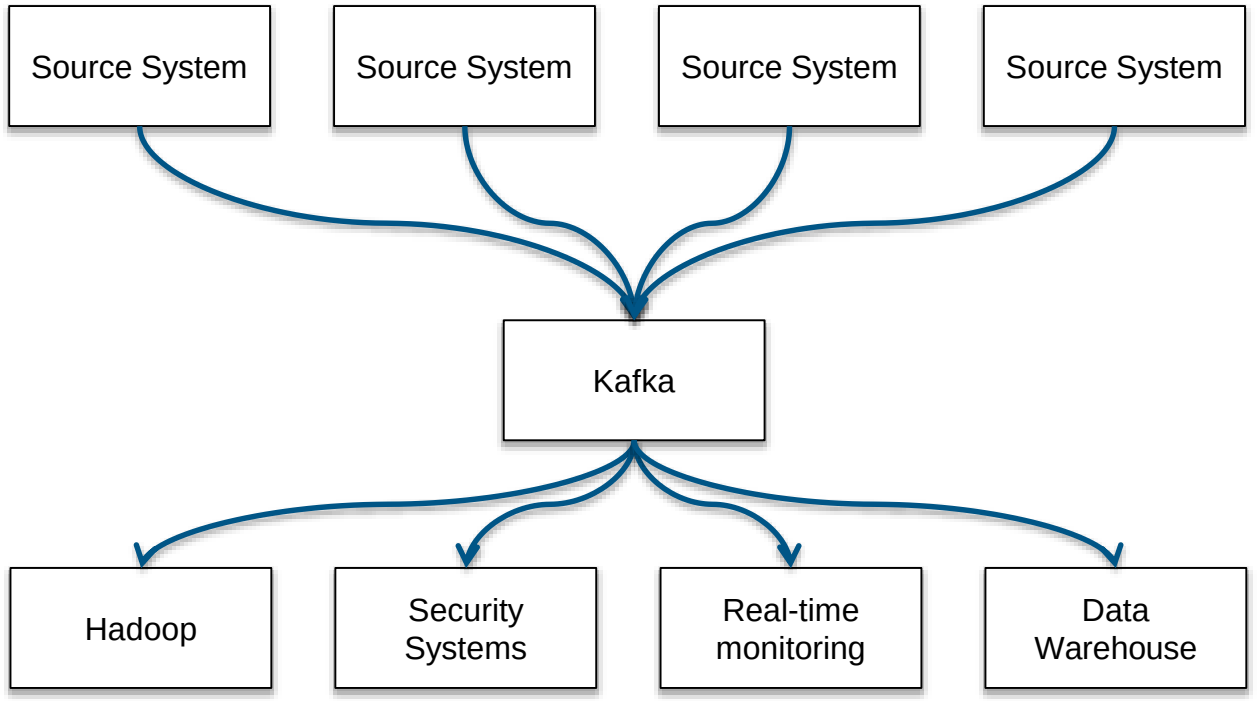
As distributed systems and services increasingly become part of a modern architecture, this makes for a fragile system

Why Kafka

Producers

Brokers

Consumers



Kafka decouples Data Pipelines

Key terminology

- Kafka maintains feeds of messages in categories called ***topics***.
- Processes that publish messages to a Kafka topic are called ***producers***.
- Processes that subscribe to topics and process the feed of published messages are called ***consumers***.
- Kafka is run as a cluster comprised of one or more servers each of which is called a ***broker***.
- Communication between all components is done via a high performance simple binary API over TCP protocol

Efficiency

- Kafka achieves it's high throughput and low latency primarily from two key concepts
- 1) Batching of individual messages to amortize network overhead and append/consume chunks together
- 2) Zero copy I/O using sendfile (Java's NIO FileChannel transferTo method).
  - Implements linux sendfile() system call which skips unnecessary copies
  - Heavily relies on Linux PageCache
    - The I/O scheduler will batch together consecutive small writes into bigger physical writes which improves throughput.
    - The I/O scheduler will attempt to re-sequence writes to minimize movement of the disk head which improves throughput.
    - It automatically uses all the free memory on the machine

Efficiency - Implication

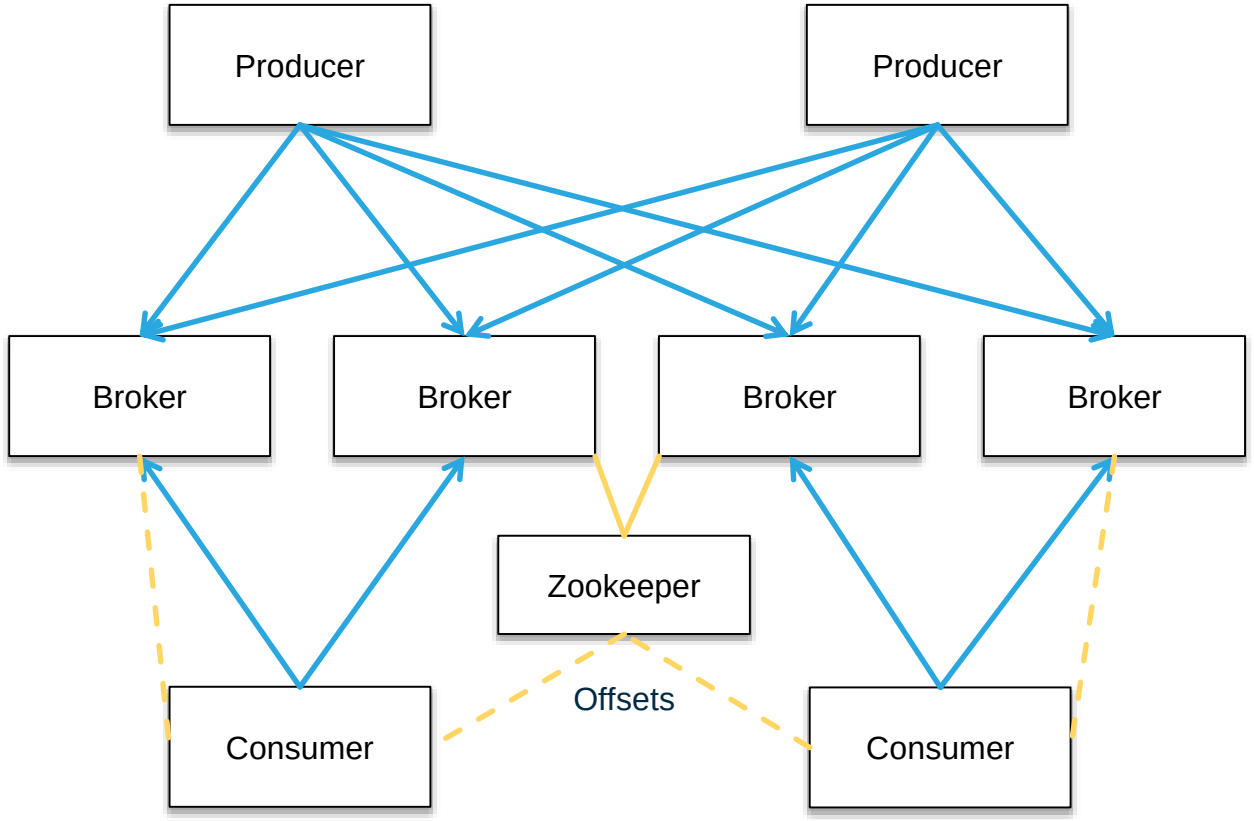
- In a system where consumers are roughly caught up with producers, you are essentially reading data from cache.
- This is what allows end-to-end latency to be so low

Architecture

Producers

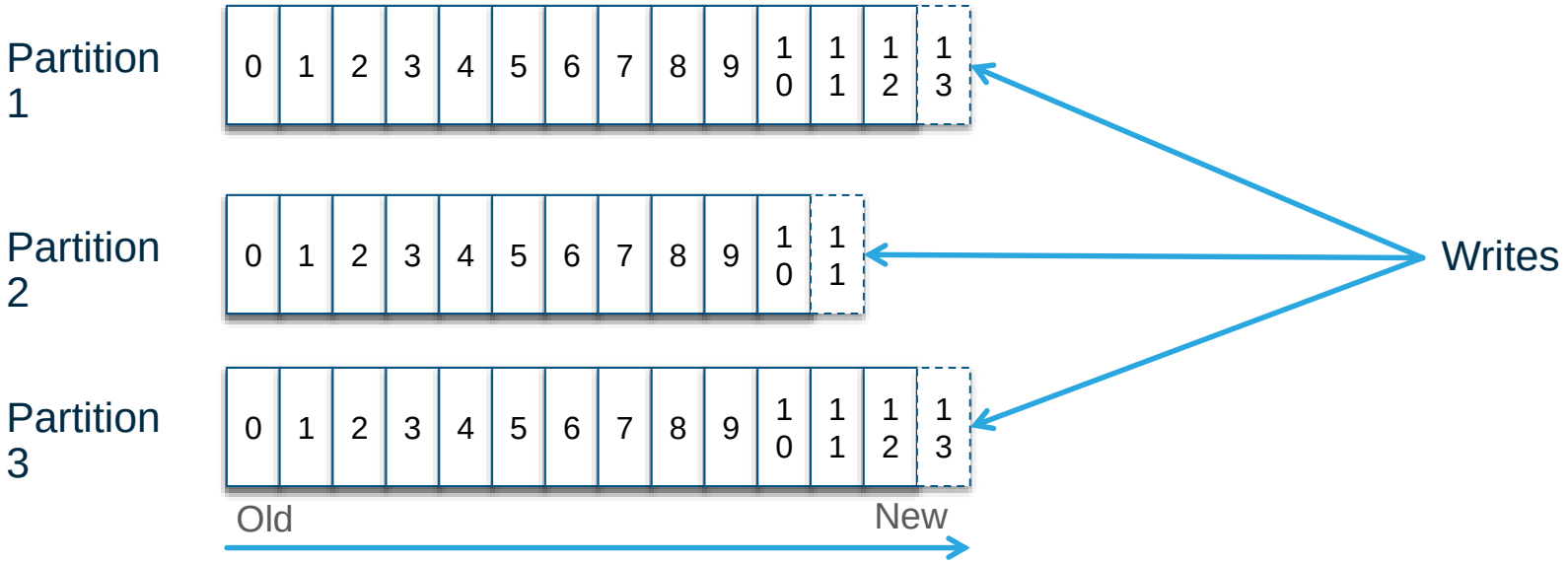
Kafka Cluster

Consumers



Topics - Partitions

- Topics are broken up into ordered commit logs called partitions.
- Each message in a partition is assigned a sequential id called an offset.
- Data is retained for a configurable period of time*



Message Ordering

- Ordering is only guaranteed within a partition for a topic
- To ensure ordering:
  - Group messages in a partition by key (producer)
  - Configure exactly one consumer instance per partition within a consumer group

Guarantees

- Messages sent by a producer to a particular topic partition will be appended in the order they are sent
- A consumer instance sees messages in the order they are stored in the log
- For a topic with replication factor N, Kafka can tolerate up to N-1 server failures without “losing” any messages committed to the log

Topics - Replication

- Topics can (and should) be replicated.
- The unit of replication is the partition
- Each partition in a topic has 1 leader and 0 or more replicas.
- A replica is deemed to be “in-sync” if
  - The replica can communicate with Zookeeper
  - The replica is not “too far” behind the leader (configurable)
- The group of in-sync replicas for a partition is called the ***ISR*** (In-Sync Replicas)
- The Replication factor cannot be lowered

Topics - Replication

- Durability can be configured with the producer configuration ***request.required.acks***
  - ***0*** The producer never waits for an ack
  - ***1*** The producer gets an ack after the leader replica has received the data
  - ***-1*** The producer gets an ack after all ISRs receive the data
- Minimum available ISR can also be configured such that an error is returned if enough replicas are not available to replicate data

Durable Writes

- Producers can choose to *trade* throughput for durability of writes:

Durability	Behaviour	Per Event Latency	Required Acknowledgements (request.required.acks)
Highest	ACK all ISRs have received	Highest	-1
Medium	ACK once the leader has received	Medium	1
Lowest	No ACKs required	Lowest	0

- Throughput can also be raised with ***more brokers***… (so do this instead)!
- A sane configuration:

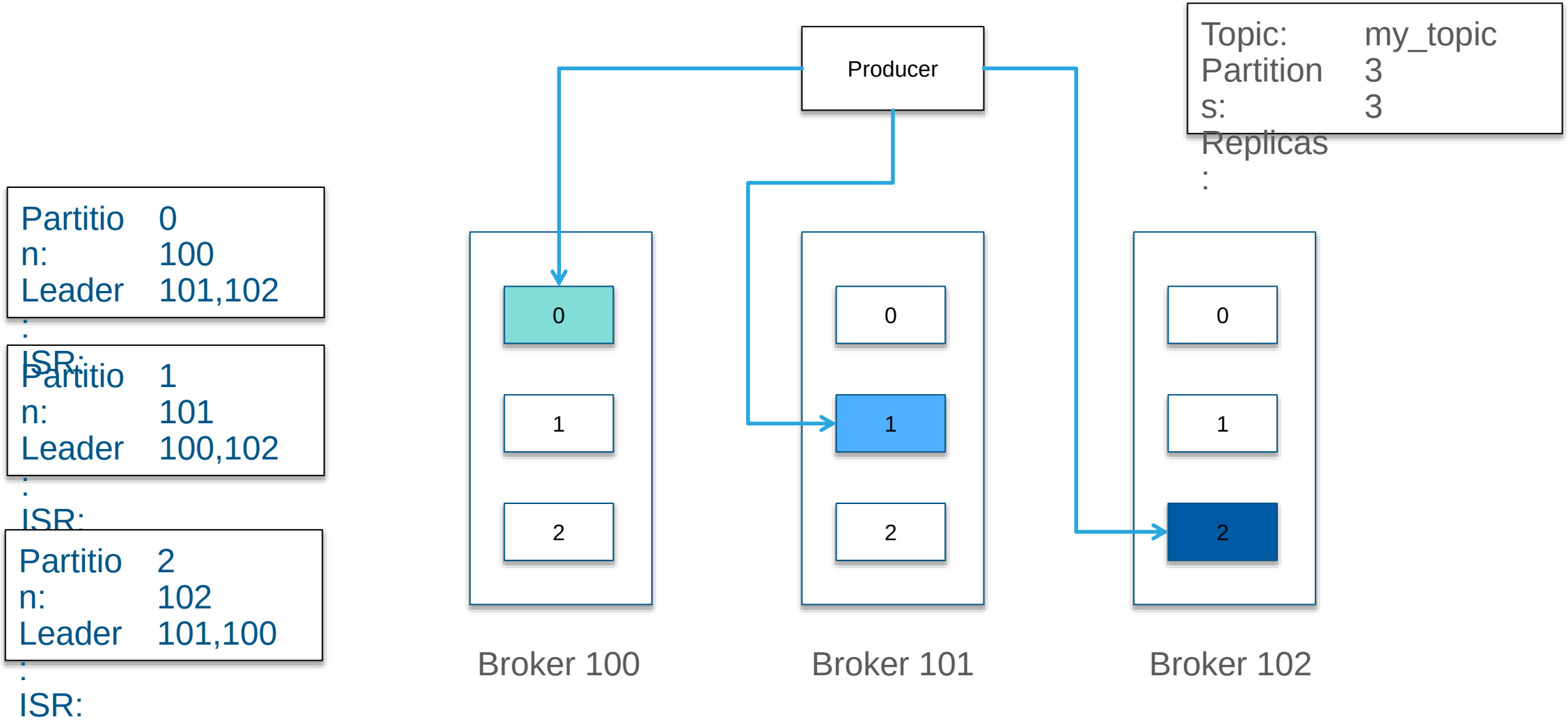
Property	Value
replication	3
min.insync.replicas	2
request.required.acks	-1

cloudera

Producer

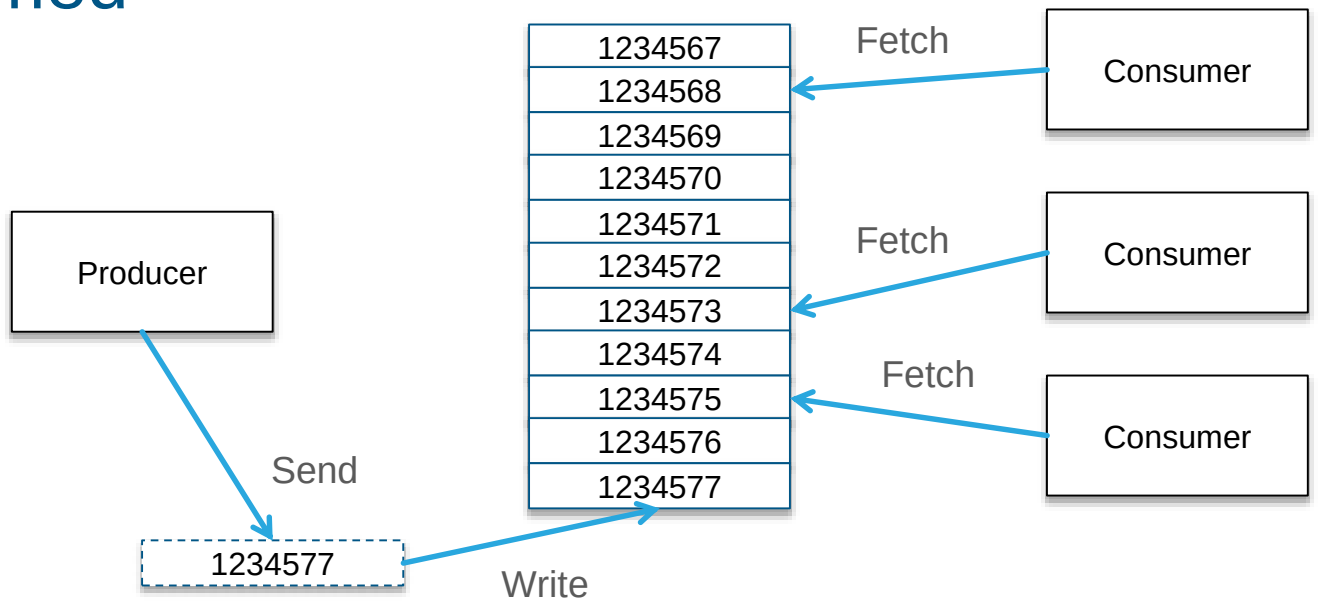
- Producers publish to a topic of their choosing (push)
- Load can be distributed
  - Typically by “round-robin”
  - Can also do “semantic partitioning” based on a key in the message
- Brokers load balance by partition
- Can support async (less durable) sending
- All nodes can answer metadata requests about:
  - Which servers are alive
  - Where leaders are for the partitions of a topic

Producer – Load Balancing and ISRs



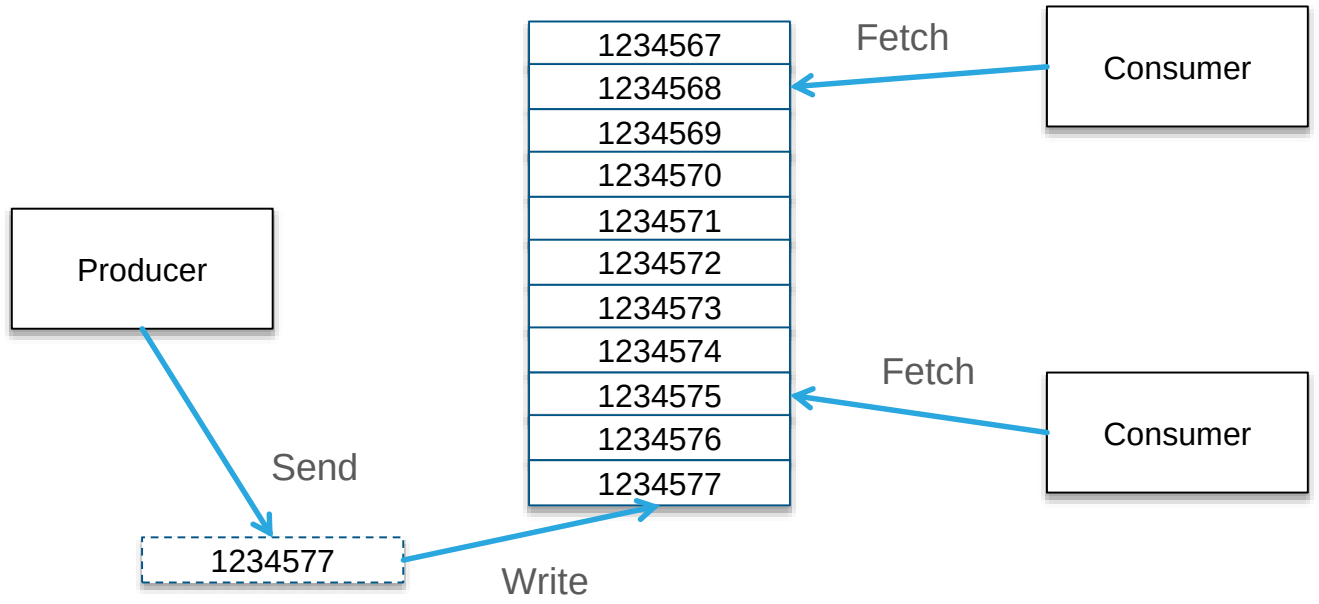
Consumer

- Multiple Consumers can read from the same topic
- Each Consumer is responsible for managing it's own offset
- Messages stay on Kafka…they are not removed after they are consumed



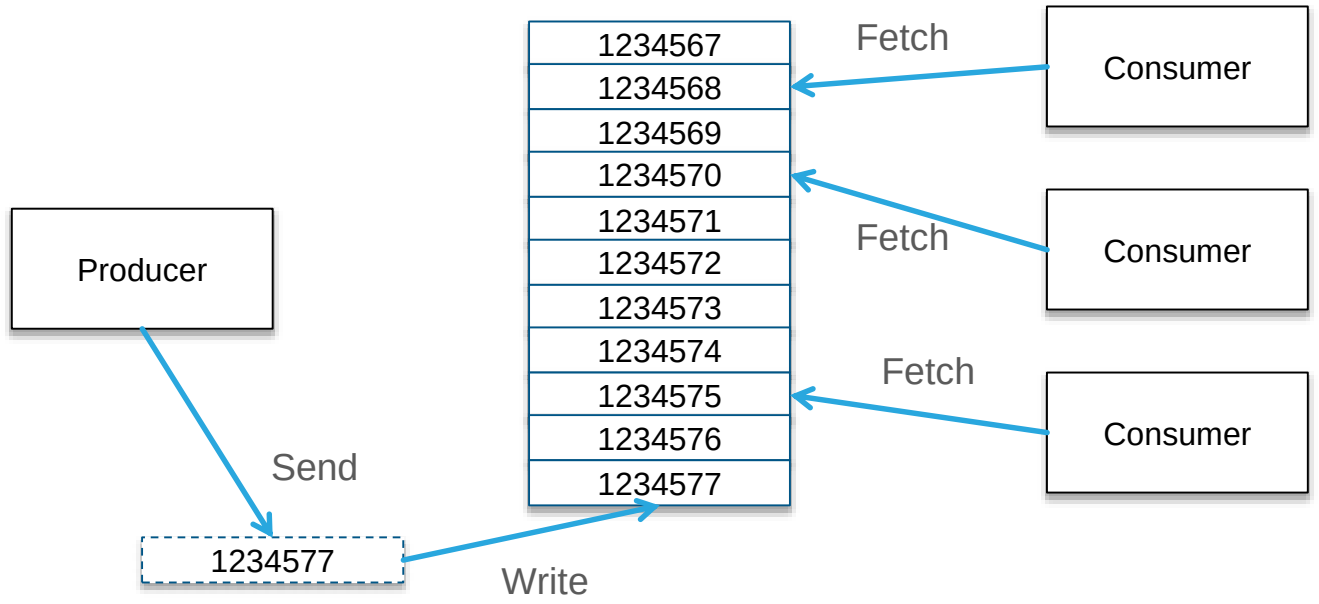
Consumer

- Consumers can go away



Consumer

- And then come back



Consumer - Groups

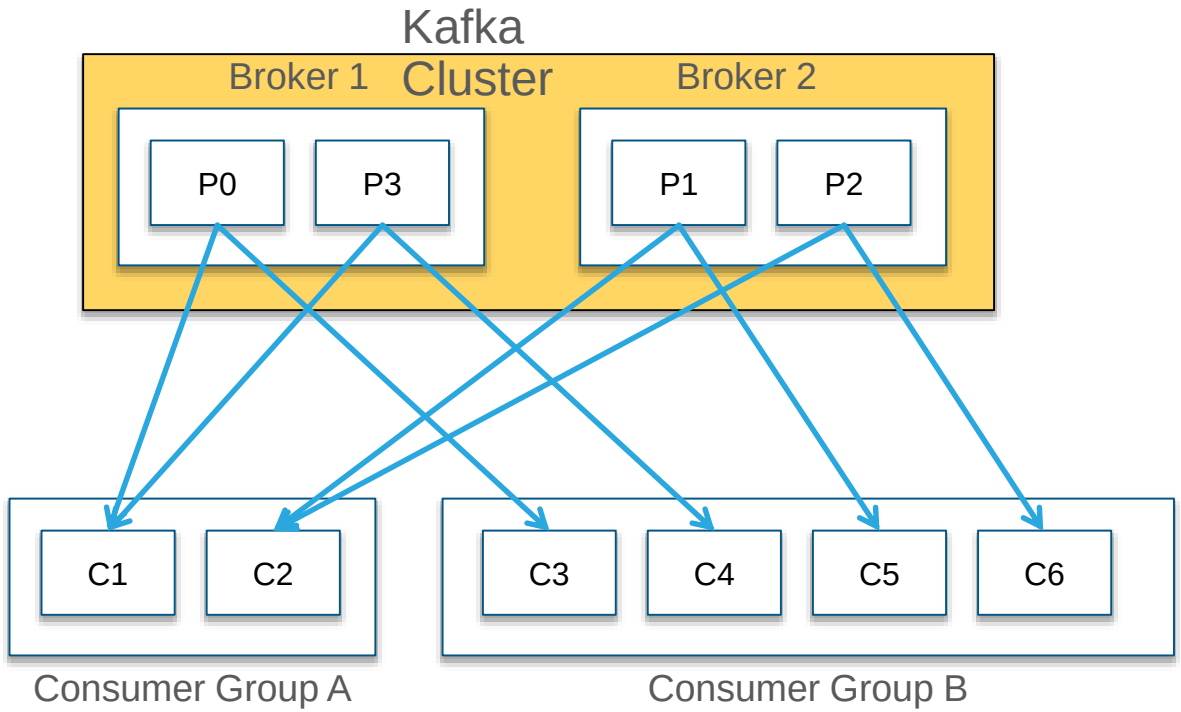
- Consumers can be organized into Consumer Groups

Common Patterns:

- 1) All consumer instances in one group
  - Acts like a traditional queue with load balancing
- 2) All consumer instances in different groups
  - All messages are broadcast to all consumer instances
- 3) “Logical Subscriber” – Many consumer instances in a group
  - Consumers are added for scalability and fault tolerance
  - Each consumer instance reads from one or more partitions for a topic
  - There cannot be more consumer instances than partitions

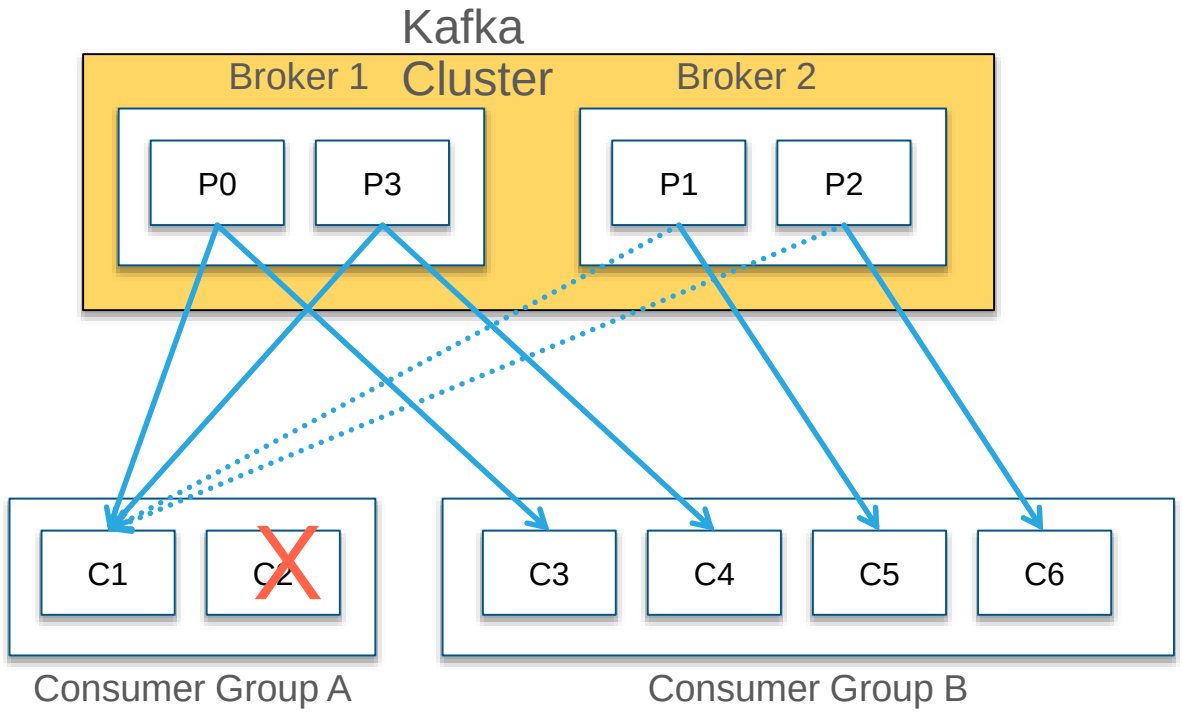
Consumer - Groups

Consumer Groups provide isolation to topics and partitions



Consumer - Groups

Can rebalance themselves



Delivery Semantics

Default

- At least once
  - **Messages are never lost but may be redelivered**
- At most once
  - Messages are lost but never redelivered
- Exactly once
  - Messages are delivered once and only once

Delivery Semantics

- At least once
  - **Messages are never lost but may be redelivered**
- At most once
  - Messages are lost but never redelivered
- Exactly once
  - Messages are delivered once and only once

Much Harder
(Impossible??)

Getting Exactly Once Semantics

- Must consider two components
  - Durability guarantees when *publishing* a message
  - Durability guarantees when *consuming* a message
- Producer
  - What happens when a produce request was sent but a network error returned before an ack?
  - Use a single writer per partition and check the latest committed value after network errors
- Consumer
  - Include a unique ID (e.g. UUID) and de-duplicate.
  - Consider storing offsets with data

Use Cases

- Real-Time Stream Processing (combined with Spark Streaming)
- General purpose Message Bus
- Collecting User Activity Data
- Collecting Operational Metrics from applications, servers or devices
- Log Aggregation
- Change Data Capture
- Commit Log for distributed systems

Positioning

Should I use Kafka ?

- For really large file transfers?
  - Probably not, it's designed for "messages" not really for files. If you need to ship large files, consider good-ole-file transfer, or breaking up the files and reading per line to move to Kafka.
- As a replacement for MQ/Rabbit/Tibco
  - Probably. Performance Numbers are drastically superior. Also gives the ability for transient consumers. Handles failures pretty well.
- If security on the broker and across the wire is important?
  - Not right now. We can't really enforce much in the way of security. (KAFKA-1682)
- To do transformations of data?
  - Not really by itself

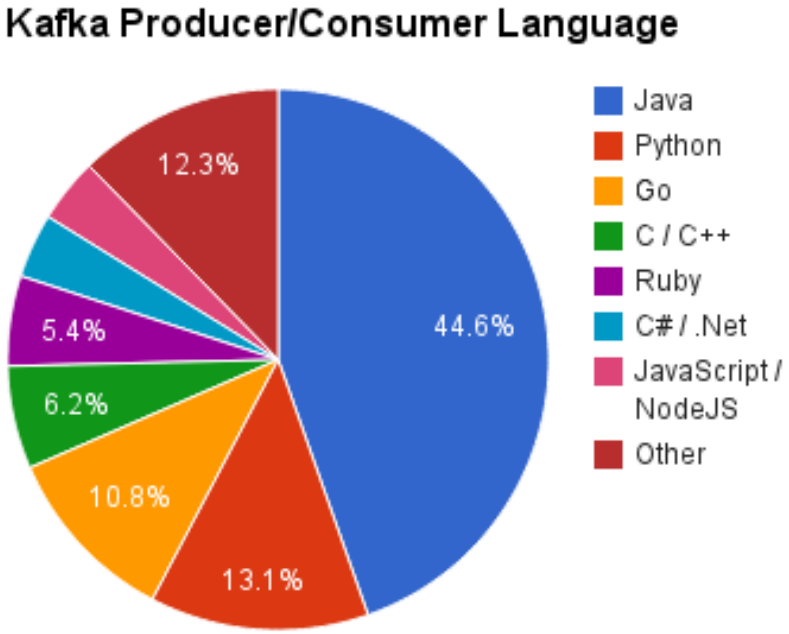
Developing with Kafka

API and Clients

cloudera

Kafka Clients

- Remember Kafka implements a binary TCP protocol.
- All clients except the JVM-based clients are maintained external to the code base.
- Full Client List here:



Java Producer Example – Old (< 0.8.1)

```
 /* start the producer */
private void start() {
producer = new Producer<String, String]]>(config);
}

/* create record and send to Kafka
* because the key is null, data will be sent to a random partition.
* the producer will switch to a different random partition every 10 minutes

**/
private void produce(String s) {
KeyedMessage<String, String]]> message = new
KeyedMessage<String, String]]>(topic, null, s);
producer.send(message);
}
```

Producer - New

```
    /**
     * Send the given record asynchronously and return a future which will eventually contain
the response information.
     *
     * @param record The record to send
     * @return A future which will eventually contain the response information
     */
    public Future send(ProducerRecord record);

    /**
     * Send a record and invoke the given callback when the record has been acknowledged
by the server
     */
    public Future send(ProducerRecord record, Callback callback);
}
```

```
// configure
Properties config = new Properties();
config.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
KafkaProducer producer = new KafkaProducer(config);

// create and send a record
ProducerRecord record = new ProducerRecord("topic", "key".getBytes(), "value".getBytes());
Future<RecordMetadata> response = producer.send(record); // this is always non-blocking
System.out.println("The offset was: " + response.get().offset()); // get() blocks
```

KAFKA-1982

```
23
24 public class Producer extends Thread
25 {
26   private final kafka.javaapi.producer.Producer<Integer, String> producer;
27   private final String topic;
28   private final Properties props = new Properties();
29
30   public Producer(String topic)
31   {
32     props.put("serializer.class", "kafka.serializer.StringEncoder");
33     props.put("metadata.broker.list", "localhost:9092");
34     // Use random partitioner. Don't need the key type. Just set it to Integer.
35     // The message is of type String.
36     producer = new kafka.javaapi.producer.Producer<Integer, String>(new ProducerConfig(props));
37     this.topic = topic;
38   }
39
40   public void run() {
41     int messageNo = 1;
42     while(true)
43     {
44       String messageStr = new String("Message_" + messageNo);
45       producer.send(new KeyedMessage<Integer, String>(topic, messageStr));
46       messageNo++;
47     }
48   }
```

```
27
28 public class Producer extends Thread
29 {
30   private final KafkaProducer<Integer, String> producer;
31   private final String topic;
32   private final Boolean isAsync;
33
34   public Producer(String topic, Boolean isAsync)
35   {
36     Properties props = new Properties();
37     props.put("bootstrap.servers", "localhost:9092");
38     props.put("client.id", "DemoProducer");
39     props.put("key.serializer", "org.apache.kafka.common.serialization.IntegerSerializer");
40     props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
41     producer = new KafkaProducer<Integer, String>(props);
42     this.topic = topic;
43     this.isAsync = isAsync;
44   }
45
46   public void run() {
47     int messageNo = 1;
48     while(true)
49     {
50       String messageStr = "Message_" + messageNo;
51       long startTime = System.currentTimeMillis();
52       if (isAsync) { // Send asynchronously
53         producer.send(new ProducerRecord<Integer, String>(topic,
54             messageNo,
55             messageStr), new DemoCallBack(startTime, messageNo, messageStr));
56         messageNo++;
57       } else { // Send synchronously
58         try {
59           producer.send(new ProducerRecord<Integer, String>(topic,
60               messageNo,
61               messageStr)).get();
62           System.out.println("Sent message: (" + messageNo + ", " + messageStr + ")");
63         } catch (InterruptedException e) {
64           e.printStackTrace();
65         } catch (ExecutionException e) {
66           e.printStackTrace();
67         }
68 >>      messageNo++;
69 >>    }
70 >>  }
71   }
```

Consumers

- New Consumer coming later in the year.
- For now two options
  - High Level Consumer -> Much easier to code against, simpler to work with, but gives you less control, particularly around managing consumption.
  - Simple Consumer -> A lot more control, but hard to implement correctly.

https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example

  - If you can, use the High Level Consumer. Be Warned. The default behavior is to “auto-commit’ offsets…this means that offsets will be committed periodically for you. If you care about data loss, don’t do this.

    https://cwiki.apache.org/confluence/display/KAFKA/Consumer+Group+Example

    http://ingest.tips/2014/10/12/kafka-high-level-consumer-frequently-missing-pieces/

Kafka Clients

- Full list on the wiki, some examples…

Log4j Appender

https://github.com/apache/kafka/blob/0.8.1/core/src/main/scala/kafka/producer/KafkaLog4jAppender.scala

```
log4j.appender.KAFKA=kafka.producer.KafkaLog4jAppender
log4j.appender.KAFKA.layout=org.apache.log4j.PatternLayout
log4j.appender.KAFKA.layout.ConversionPattern=[%d] %p %m (%c)%n
log4j.appender.KAFKA.BrokerList=192.168.86.10:9092
log4j.appender.KAFKA.ProducerType=async
log4j.appender.KAFKA.Topic=logs
```

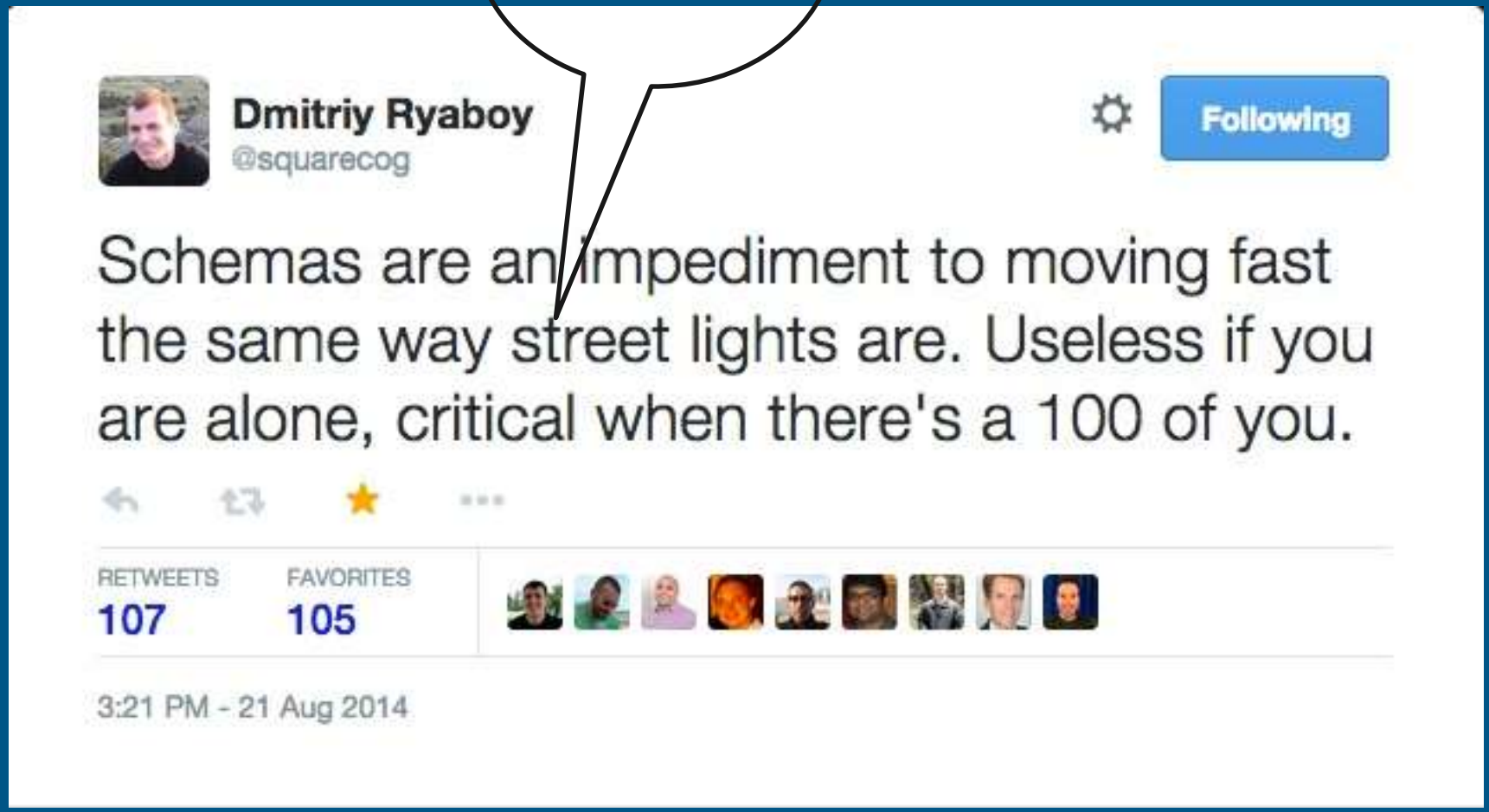
Syslog Producer

- Syslog producer - https://github.com/stealthly/go_kafka_client/tree/master/syslog

Data Formatting

- There are three basic approaches
- Wild West Crazy
- LinkedIn Way
  - https://github.com/schema-repo/schema-repo
  - http://confluent.io/docs/current/schema-registry/docs/index.html
- Parse-Magic Way – Scaling Data

He means traffic lights



Having understood the downsides of working with plaintext log messages, the next logical evolution step is to use a loosely-structured, easy-to-evolve markup scheme. JSON is a popular format we've seen adopted by multiple companies, and tried ourselves. It has the advantage of being easily parseable, arbitrarily extensible, and amenable to compression.

JSON log messages generally start out as an adequate solution, but then gradually spiral into a ***persistent nightmare***

Jimmy Lin and Dmitriy Ryaboy

Twitter, Inc

Scaling Big Data Mining Infrastructure:

The Twitter Experience

http://www.kdd.org/sites/default/files/issues/14-2-2012-12/V14-02-02-Lin.pdf

Administration

Basic

cloudera

Installation / Configuration

1. Download the Cloudera Labs Kafka CSD into /opt/cloudera/csd/
2. Restart CM

```
[root@dev ~]# cd /opt/cloudera/csd/
[root@dev csd]# wget http://archive-primary.cloudera.com/cloudera-labs/csds/kafka/CLABS_KAFKA-1.0.0.jar
--2014-11-27 14:57:07--  http://archive-primary.cloudera.com/cloudera-labs/csds/kafka/CLABS_KAFKA-1.0.0.jar
Resolving archive-primary.cloudera.com... 184.73.217.71
Connecting to archive-primary.cloudera.com|184.73.217.71|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5017 (4.9K) [application/java-archive]
Saving to: "CLABS_KAFKA-1.0.0.jar"

100%[======================================================================>] 5,017       --.-K/s   in 0s

2014-11-27 14:57:07 (421 MB/s) - "CLABS_KAFKA-1.0.0.jar" saved [5017/5017]

[root@dev csd]# service cloudera-scm-server restart
Stopping cloudera-scm-server:                              [  OK  ]
Starting cloudera-scm-server:                              [  OK  ]
[root@dev csd]#
```

3. CSD will be included in CM as of 5.4

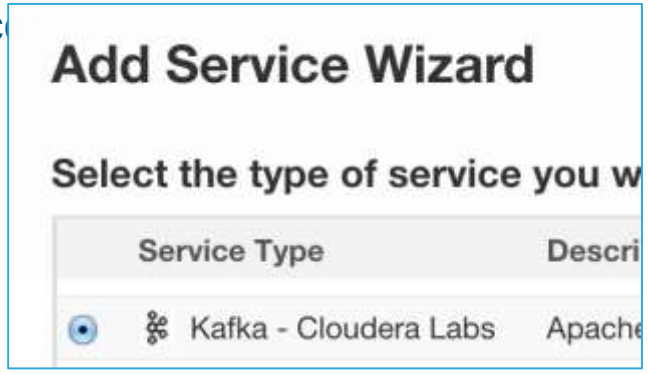
Source: http://www.cloudera.com/content/cloudera/en/documentation/cloudera-kafka/latest/PDF/cloudera-kafka.pdf

cloudera

Installation / Configuration

3. Download and distribute the Kafka parcel:

3. Add a Kafka servic



4. Review Configuration

Source: http://www.cloudera.com/content/cloudera/en/documentation/cloudera-kafka/latest/PDF/cloudera-kafka.pdf

Usage

- Create a topic*:

```
bin/kafka-topics.sh --zookeeper zkhost:2181 --create --topic foo --replication-factor 1 --partitions 1
```

- List topics:

```
bin/kafka-topics.sh --zookeeper zkhost:2181 --list
```

- Write data:

```
cat data | bin/kafka-console-producer.sh --broker-list brokerhost:9092 --topic test
```

- Read data:

```
bin/kafka-console-consumer.sh --zookeeper zkhost:2181 --topic test --from-beginning
```

* deleting a topic not possible in 0.8.2

Kafka Topics - Admin

- Commands to administer topics are via shell script:

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

bin/kafka-topics.sh --list --zookeeper localhost:2181
```

Topics can be created dynamically by producers…don't do this

Sizing and Planning

Guidelines and estimates

cloudera

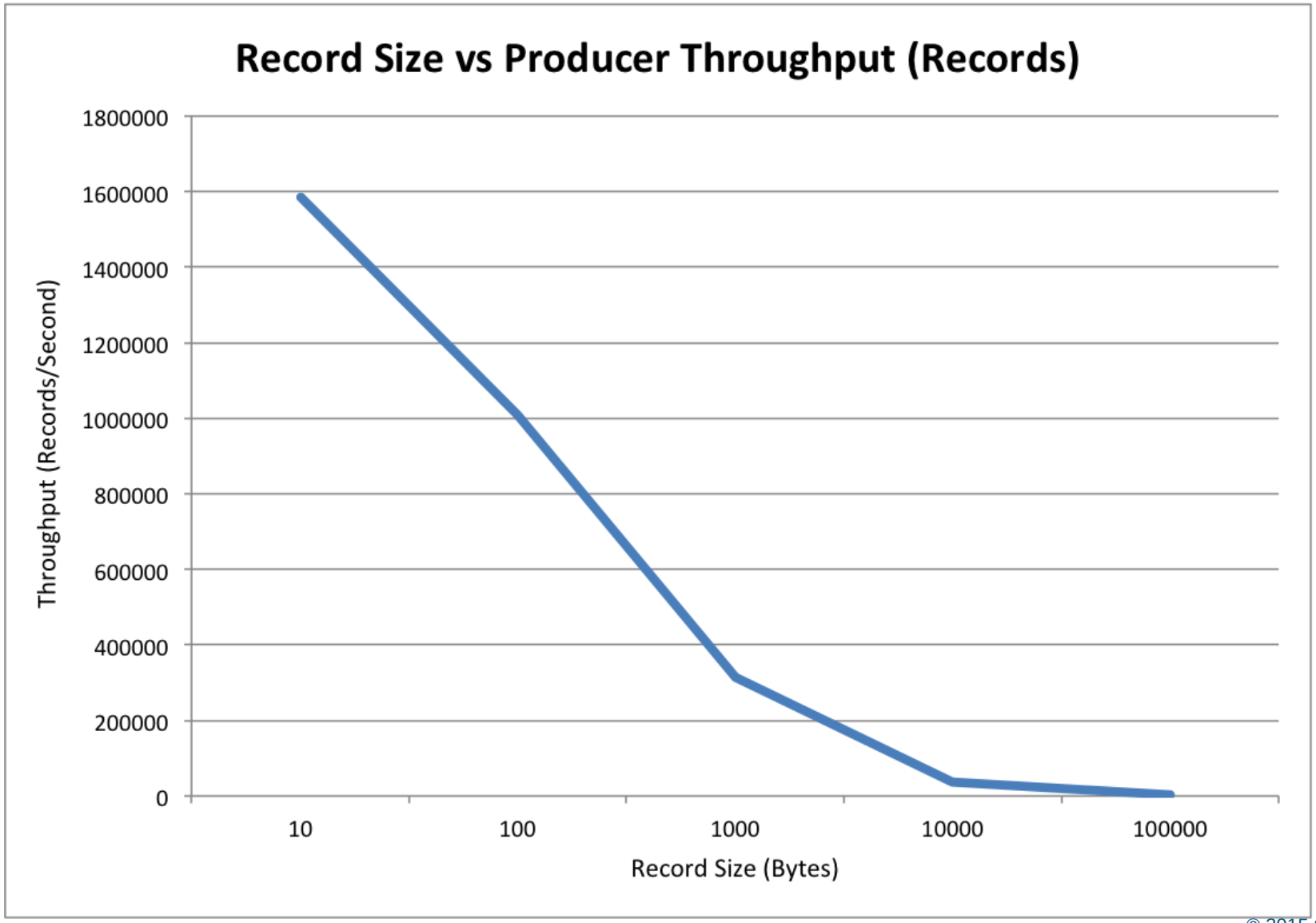
Sizing / Planning

- Cloudera is in the process of benchmarking on different hardware configurations
- Early testing shows that relatively small clusters (3-5 nodes) can process > 1M Messages/s * ----- with some caveats
- We will release these when they are ready

Producer Performance – Single Thread

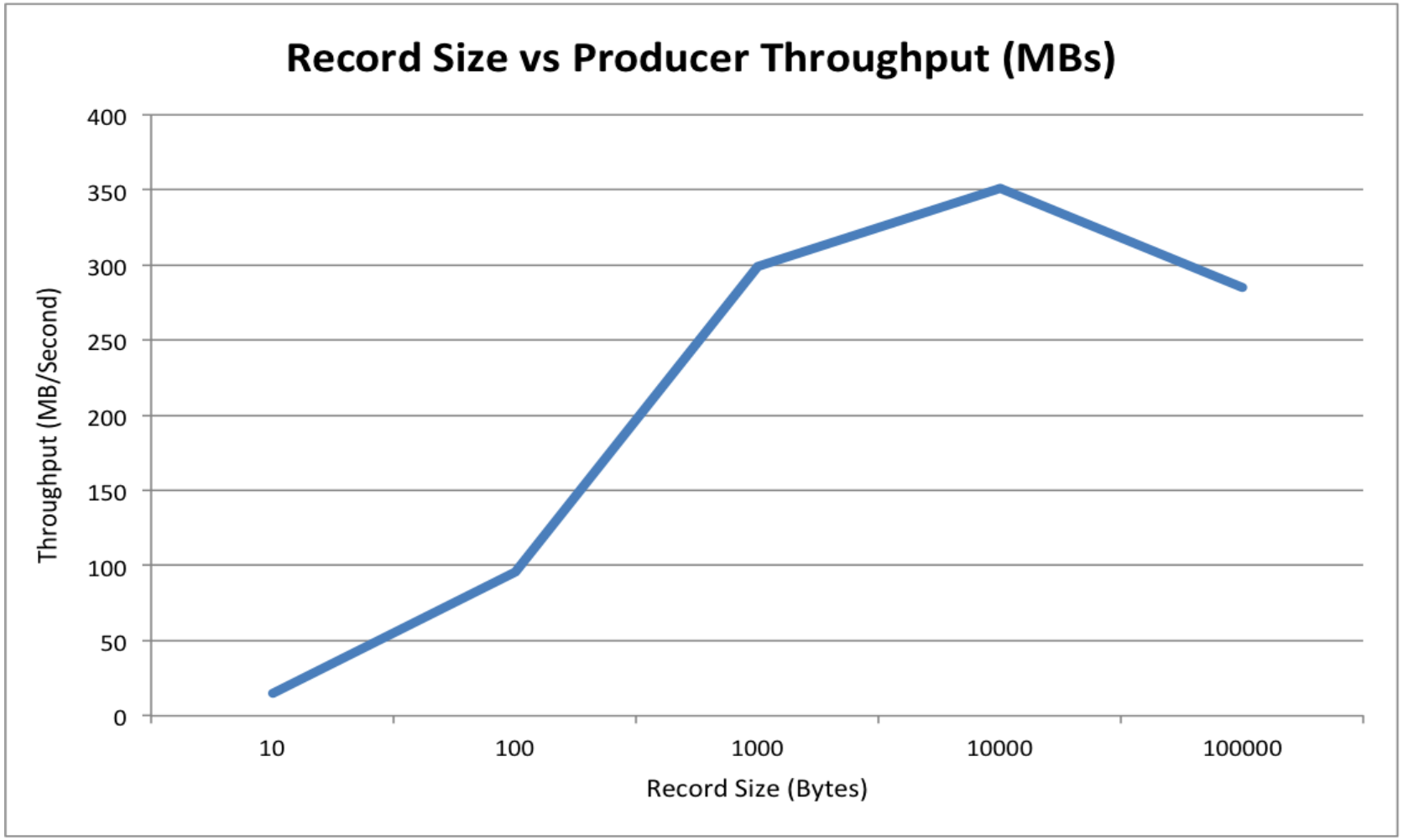
Type	Records/sec	MB/s	Avg Latency (ms)	Max Latency	Median Latency	95th %tile
No Replication	1,100,182	104	42	1070	1	362
3x Async	1,056,546	101	42	1157	2	323
3x Sync	493,855	47	379	4483	192	1692

Benchmark Results



cloudera

Benchmark Results



Hardware Selection

- A standard Hadoop datanode configuration is recommended.
- 12-14 disks is sufficient. SATA 7200 (more frequent flushes may benefit from SAS)
- RAID 10 or JBOD **--- I have an opinion about this. Not necessarily shared by all**
- Writes in JBOD will round-robin but all writes to a given partition will be contiguous. This could lead to imbalance across disks
  - Rebuilding a failed volume can effectively can effectively disable a server with I/O requests. (debatable)
  - Kafka built-in replication should make up for reliability provided by RAID
  - Kafka utilizes Page Cache heavily so additional memory will be put to good use.
- For a lower bound calculate write_throughput * time_in_buffer(s)

Other notes

- 10GB bonded NICs are preferred – With no async can definitely saturate a network
- Consider a dedicated zookeeper cluster (3-5 nodes)
  - 1U machines, 2-4 SSDs. Stock RAM –
  - This less of thing with Kafka-based offsets, you can share with Hadoop if you want

Cloudera Integration

Current state and Roadmap

cloudera

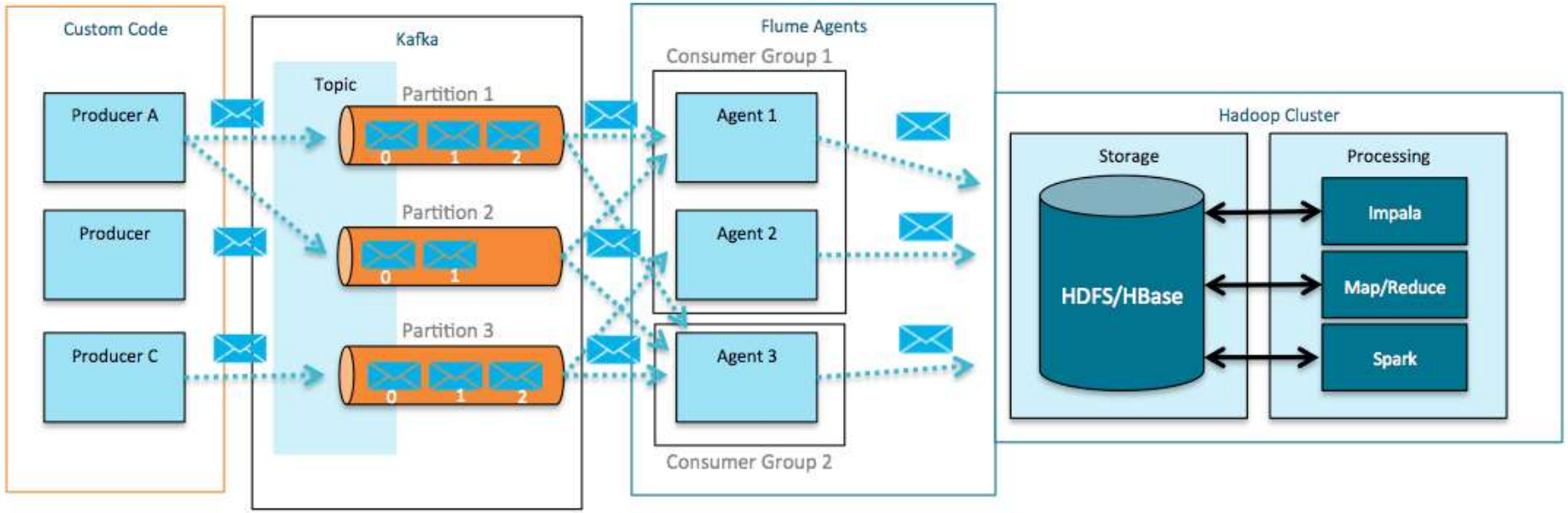
Flume (Flafka)

- Allows for zero-coding ingestion into HDFS / HBase / Solr
- Can utilize custom routing and per-event processing via selectors and interceptors
- Batch size of source can be tuned for throughput/latency
- Lots of other stuff… full details here:

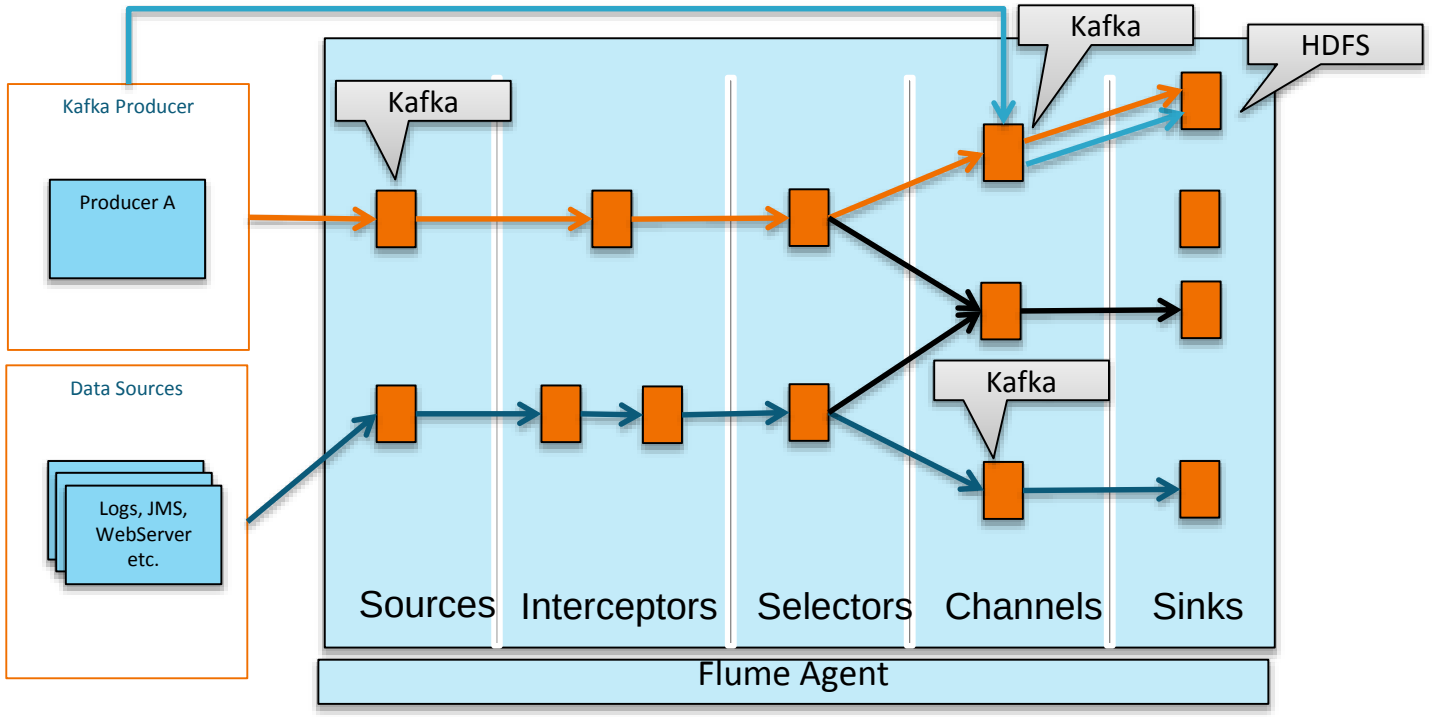
http://blog.cloudera.com/blog/2014/11/flafka-apache-flume-meets-apache-kafka-for-event-processing/

Flume (Flafka)

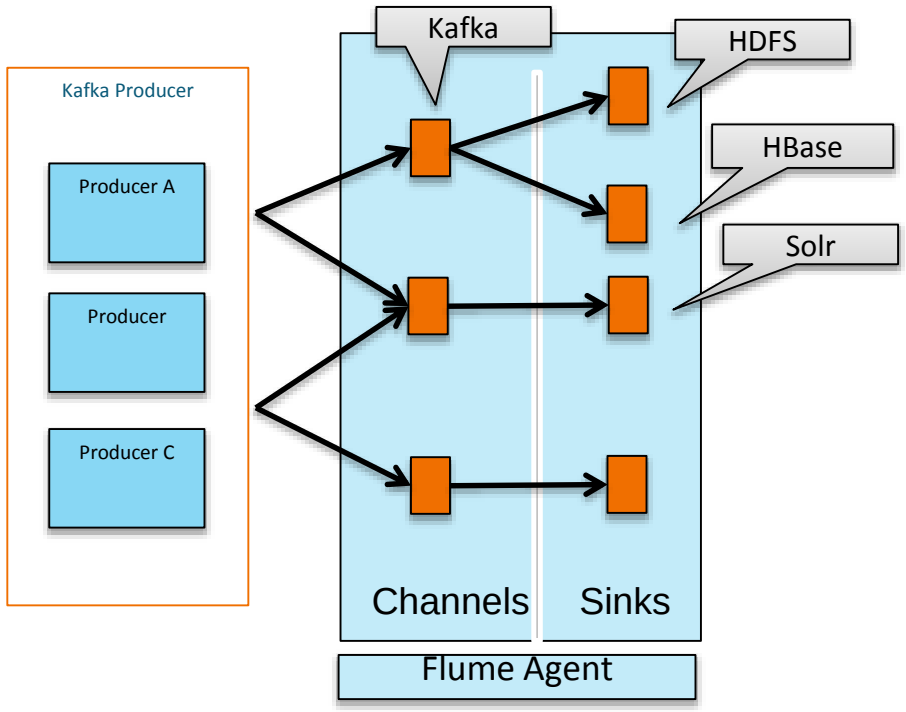
- Flume Integrates with Kafka as of CDH 5.2
  - Source
  - Sink
  - Channel



Flafka



Flafka Channel



Spark Streaming

Consuming data in parallel for high throughput:

- Each input DStream creates a single receiver, which runs on a single machine. But often in Kafka, the NIC card can be a bottleneck. To consume data in parallel from multiple machines initialize multiple KafkaInputDStreams with the same Consumer Group Id. Each input DStream will (for the most part) run on a separate machine.
- In addition, each DStream can use multiple threads to consume from Kafka

Writing to Kafka from Spark or Spark Streaming:

Directly use the Kafka Producer API from your Spark code.

Example:

KafkaWordCount

http://www.michael-noll.com/blog/2014/10/01/kafka-spark-streaming-integration-example-tutorial/

New in Spark 1.3

- http://blog.cloudera.com/blog/2015/03/exactly-once-spark-streaming-from-apache-kafka/
- The new release of Apache Spark, 1.3, includes new RDD and DStream implementations for reading data from Apache Kafka.
  - More uniform usage of Spark cluster resources when consuming from Kafka
  - Control of message delivery semantics
  - Delivery guarantees without reliance on a write-ahead log in HDFS
  - Access to message metadata

Other Streaming

- Storm
- Samza
- VoltDB (kinda)
- DataTorrent
- Tigon

Upcoming

New Stuff

cloudera

New in Current Release (0.8.2)

- New Producer (we covered)
- Delete Topic
- New offset Management(we covered)
- Automated Leader Rebalancing
- Controlled Shutdown
- Turn off Unclean Leader Election
- Min ISR (we covered)
- Connection Quotas

Slated for Next Release(s)

- New Consumer (Already in trunk)
- SSL and Security Enhancements (In review)
- New Metrics (under discussion)
- File-buffer-backed producer (maybe)
- Admin CLI (In Review)
- Enhanced Mirror Maker – Replication (
- Better docs / code cleanup
- Quotas
- Purgatory Redesign

cloudera

Thank you.