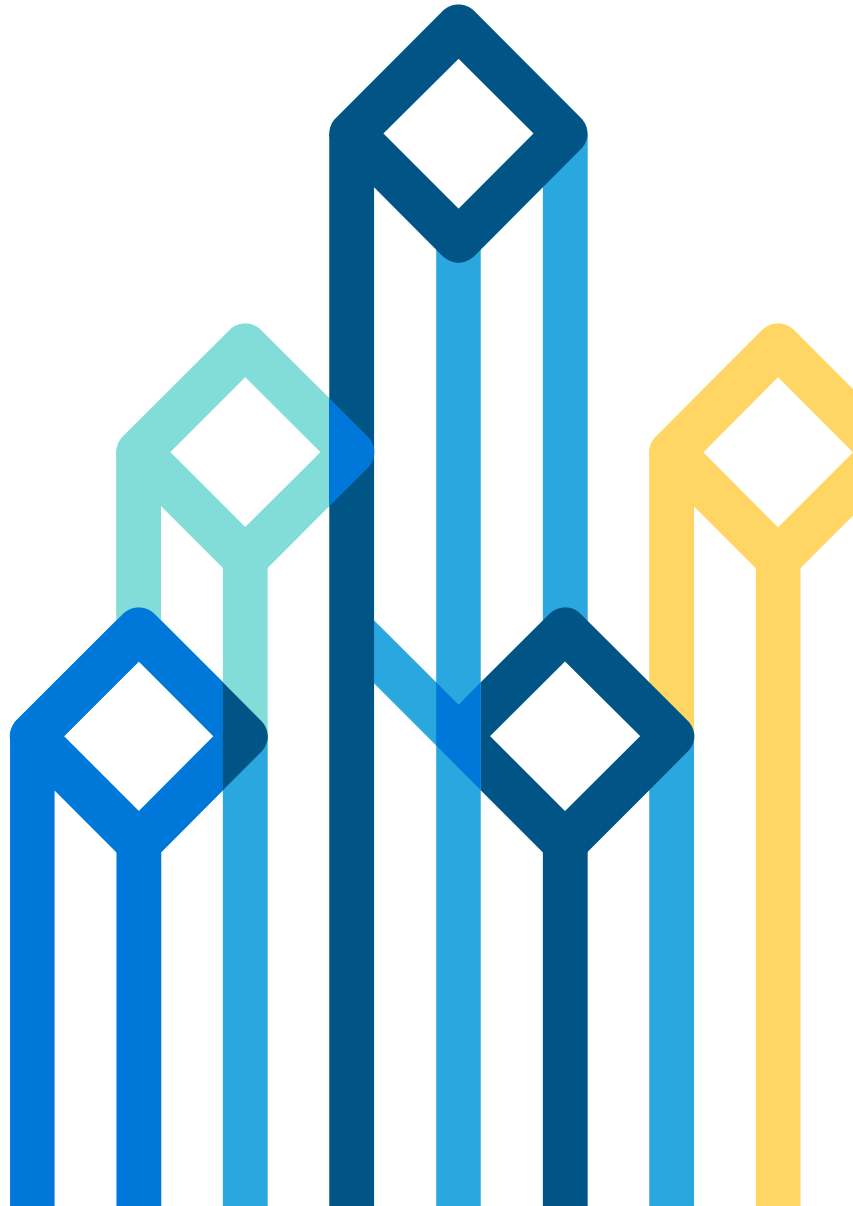
Kafka Introduction



Rahul Shukla

Agenda

- Introduction
- Concepts
- Use Cases
- Development
- Administration

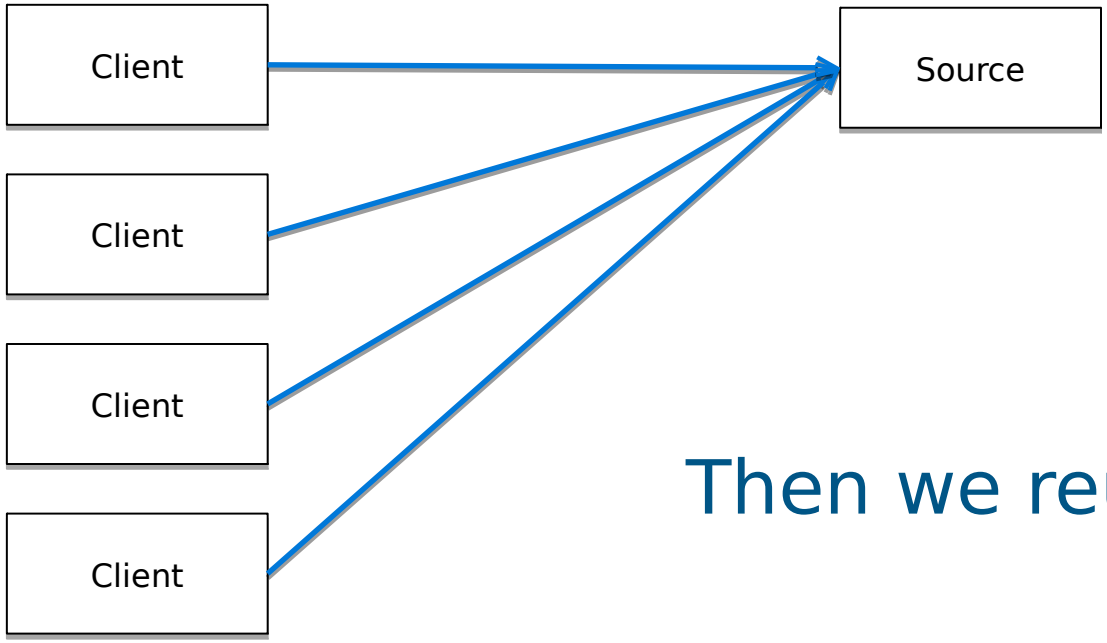
Concepts

Basic Kafka Concepts

Why Kafka

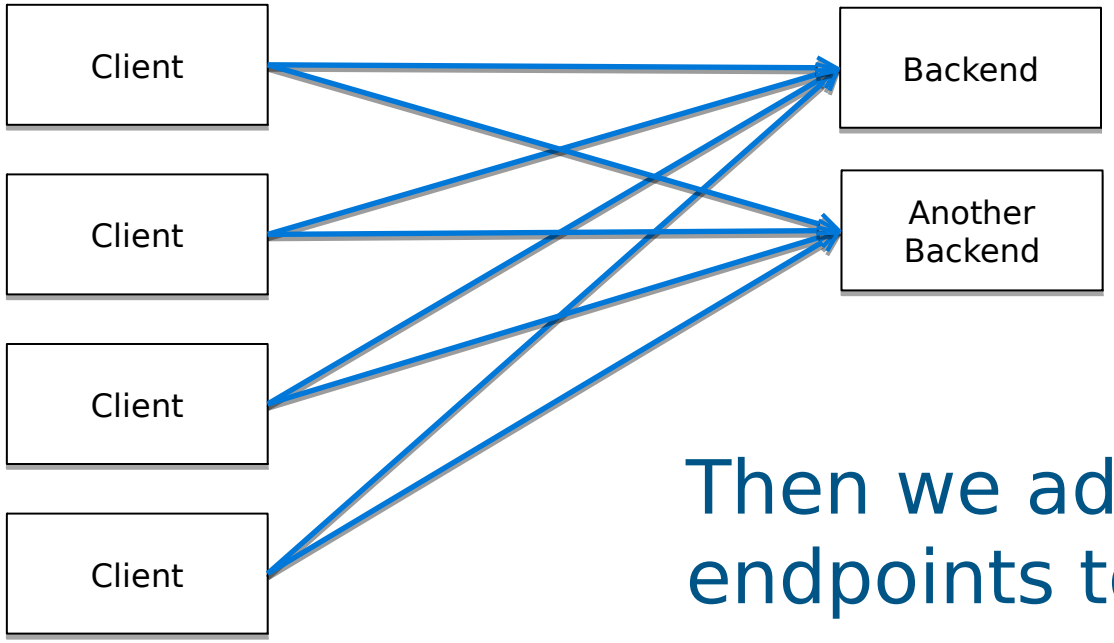
Data Pipelines Start like this.

Why Kafka



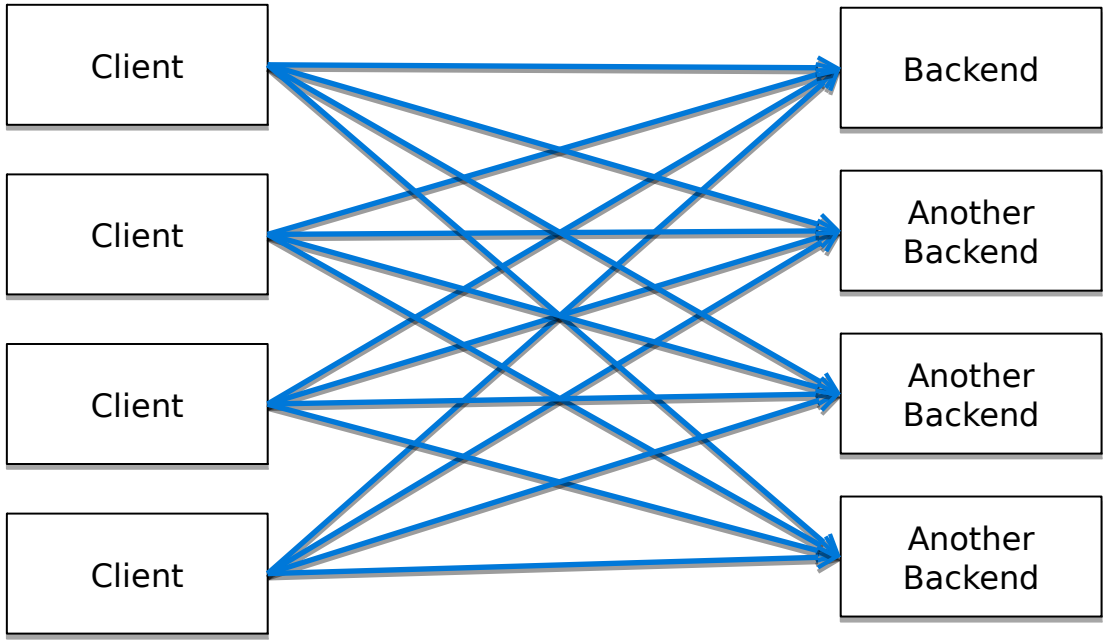
Then we reuse them

Why Kafka



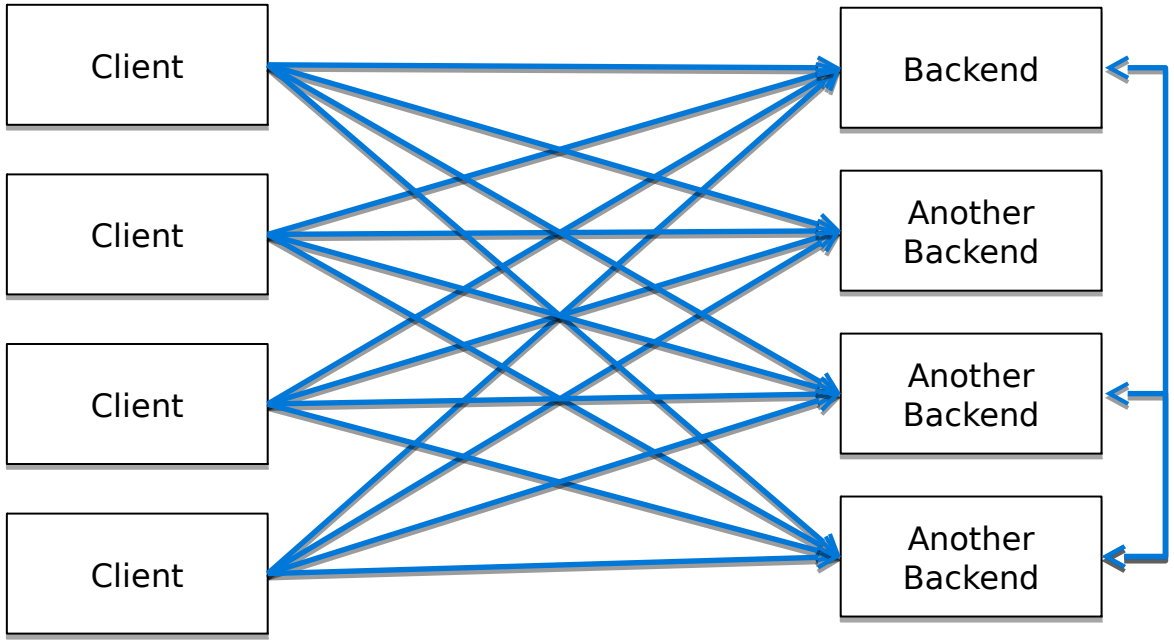
Then we add additional endpoints to the existing sources

Why Kafka



Then it starts to look like this

Why Kafka



With maybe some of this

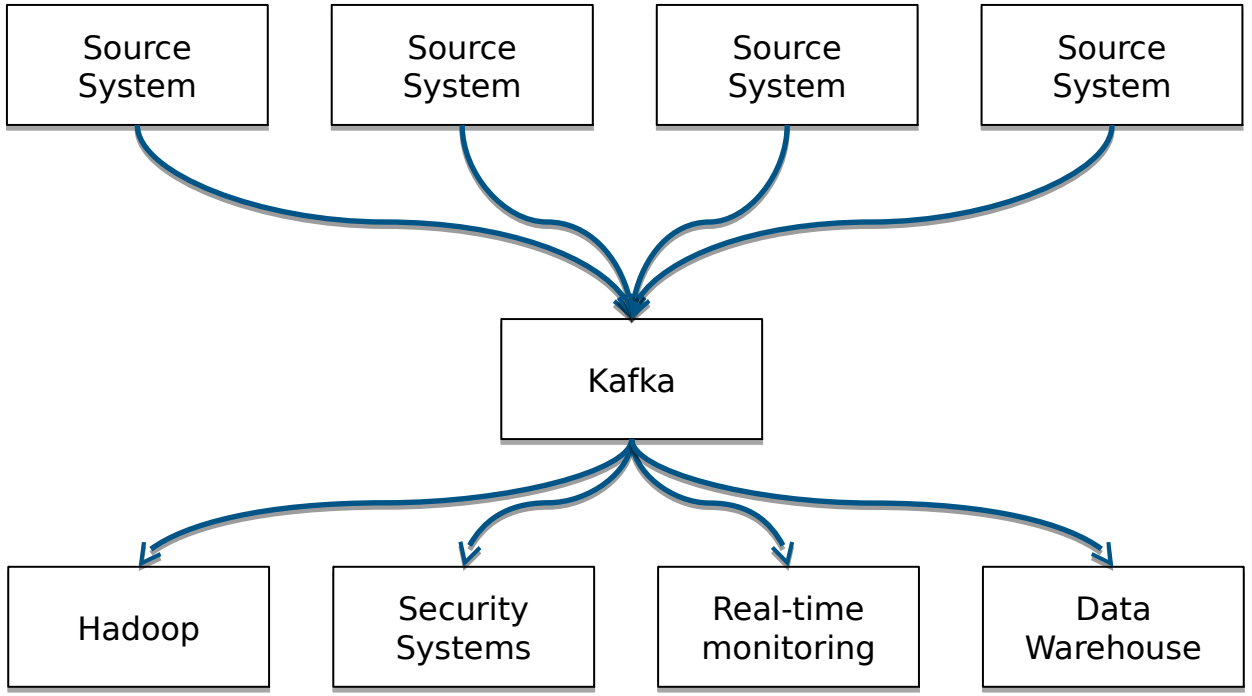
As distributed systems and services increasingly become part of a modern architecture, this makes for a fragile system

Why Kafka

Producers

Brokers

Consumers



Kafka decouples Data Pipelines

Key terminology

- Kafka is run as a cluster comprised of one or more servers each of which is called a ***broker***.
- Kafka maintains feeds of messages in categories called ***topics***.
- Kafka maintains each topic in 1 or more ***partitions***.
- 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***.
- 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

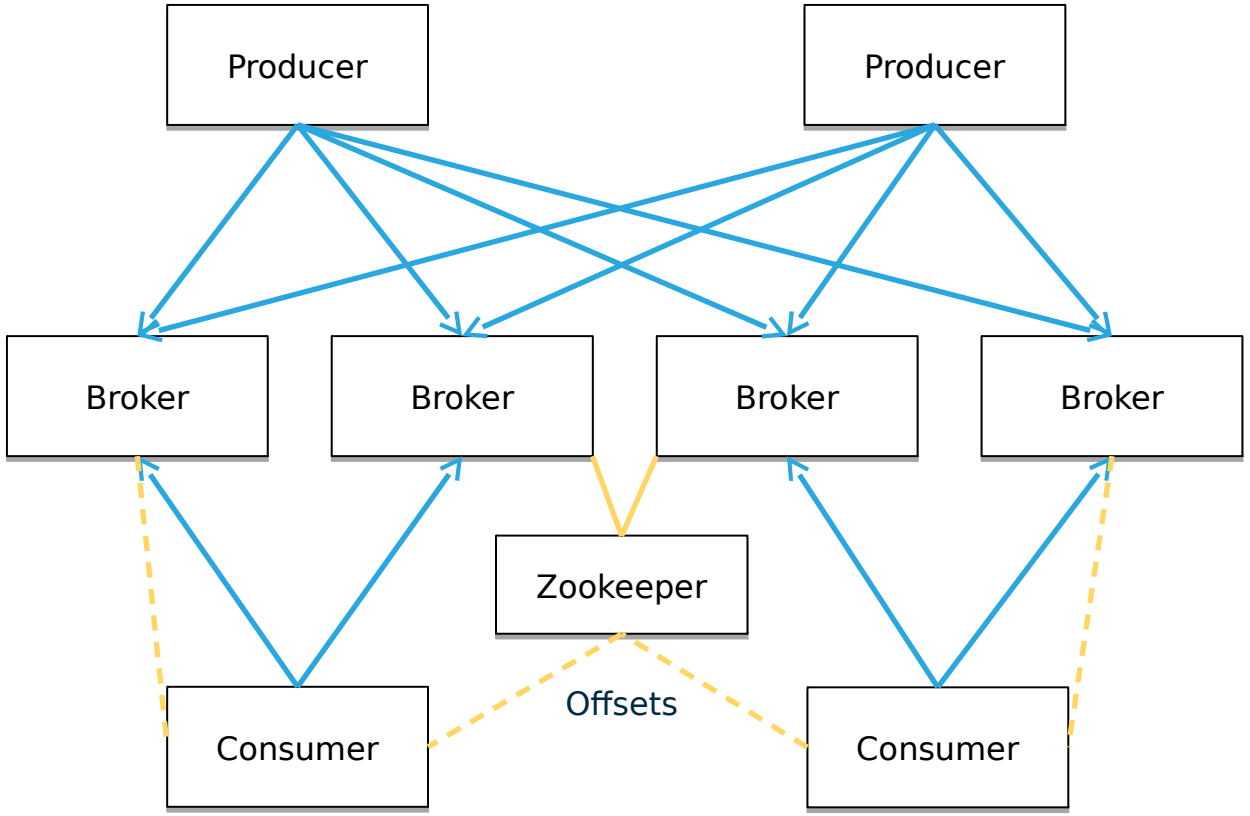
- In a system where consumers are roughly caught up with producers, you are essentially reading data from cache.
- This is what allows end-to-end latency to be so low

Architecture

Producers

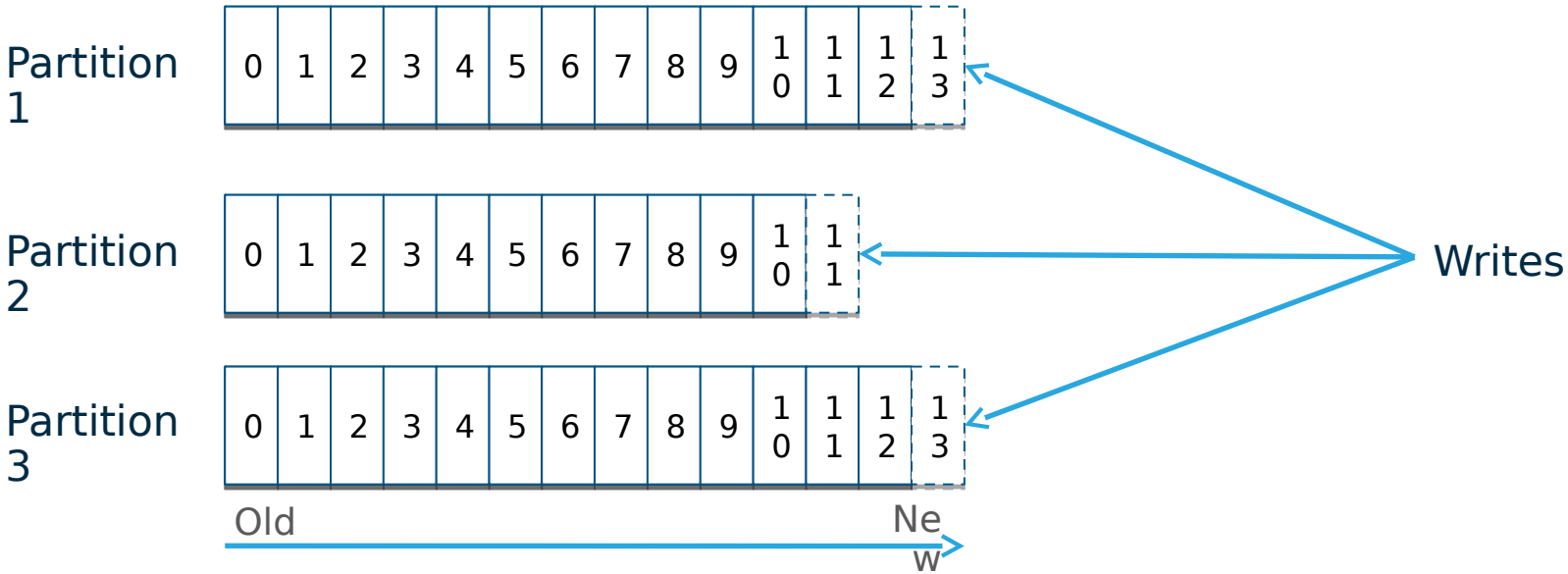
Kafka Cluster

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

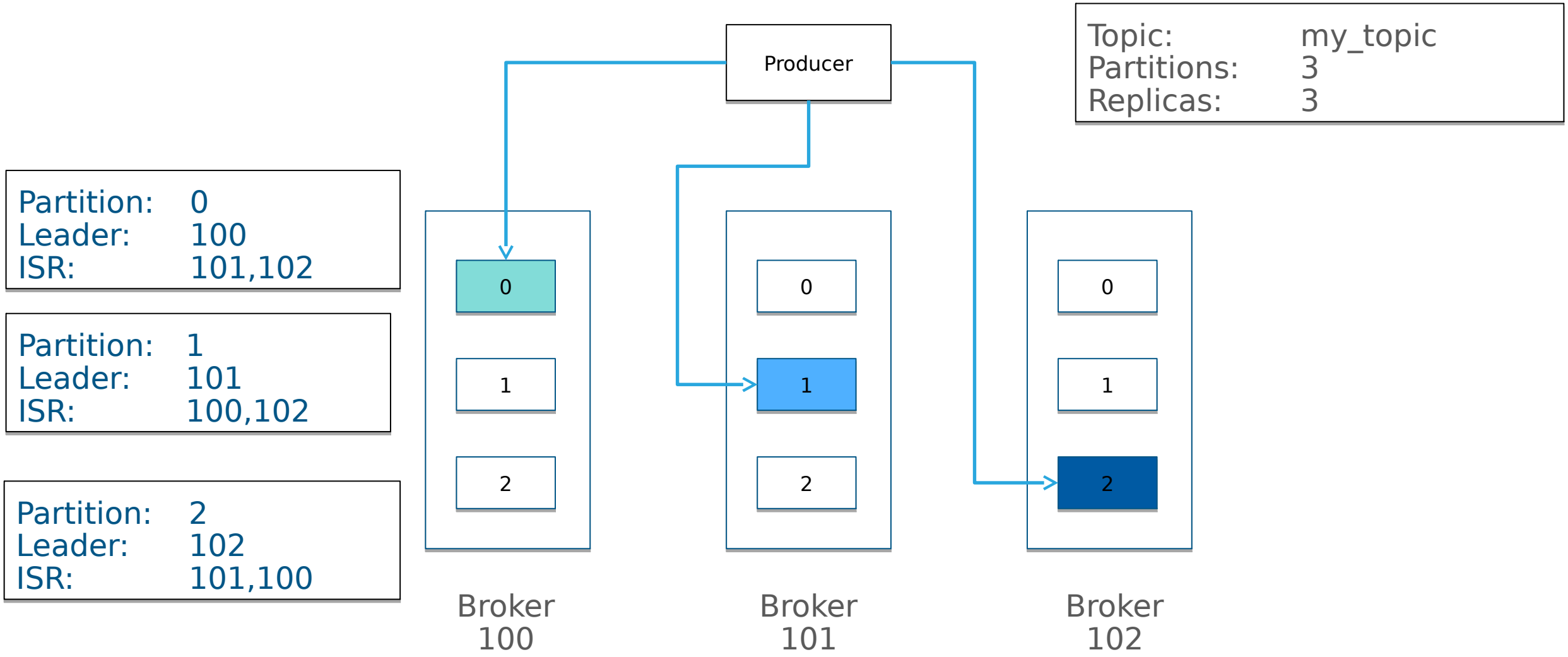
- Throughput can also be raised with ***more brokers***... (so do this instead)!
- A sane configuration:

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

Producer

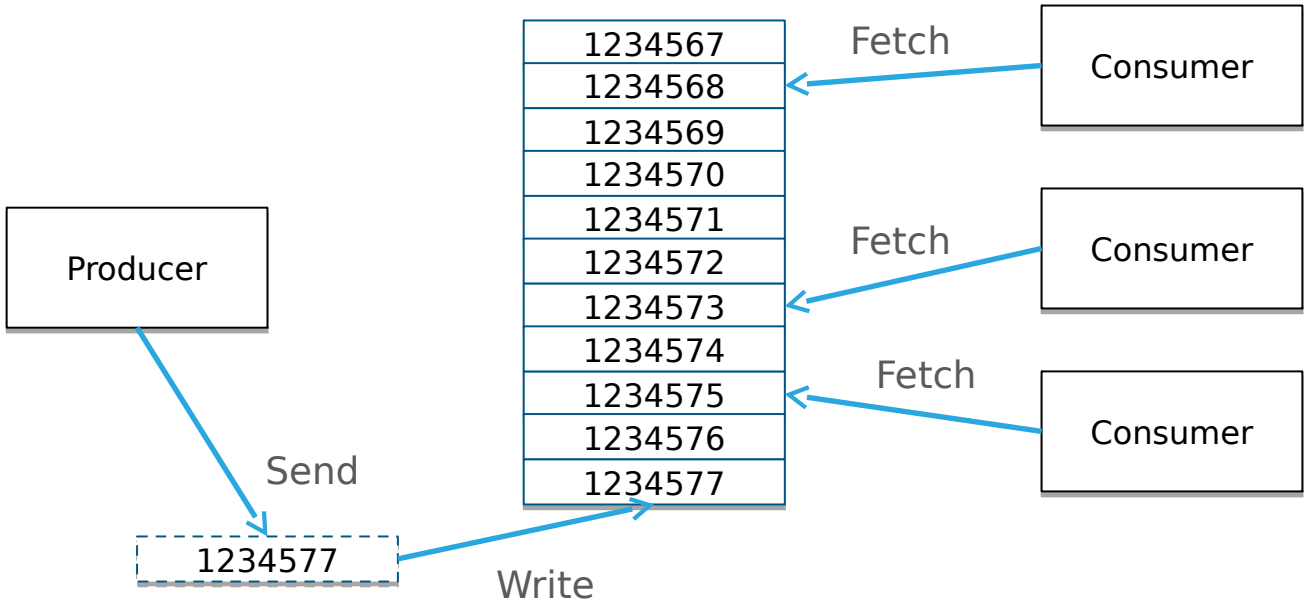
- Producers publish to a topic of their choosing (push)
- Load can be distributed
  - Typically by “round-robin”
  - Can also do “semantic partitioning” based on a key in the message
- Brokers load balance by partition
- Can support async (less durable) sending
- All nodes can answer metadata requests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

Producer – Load Balancing and ISRs



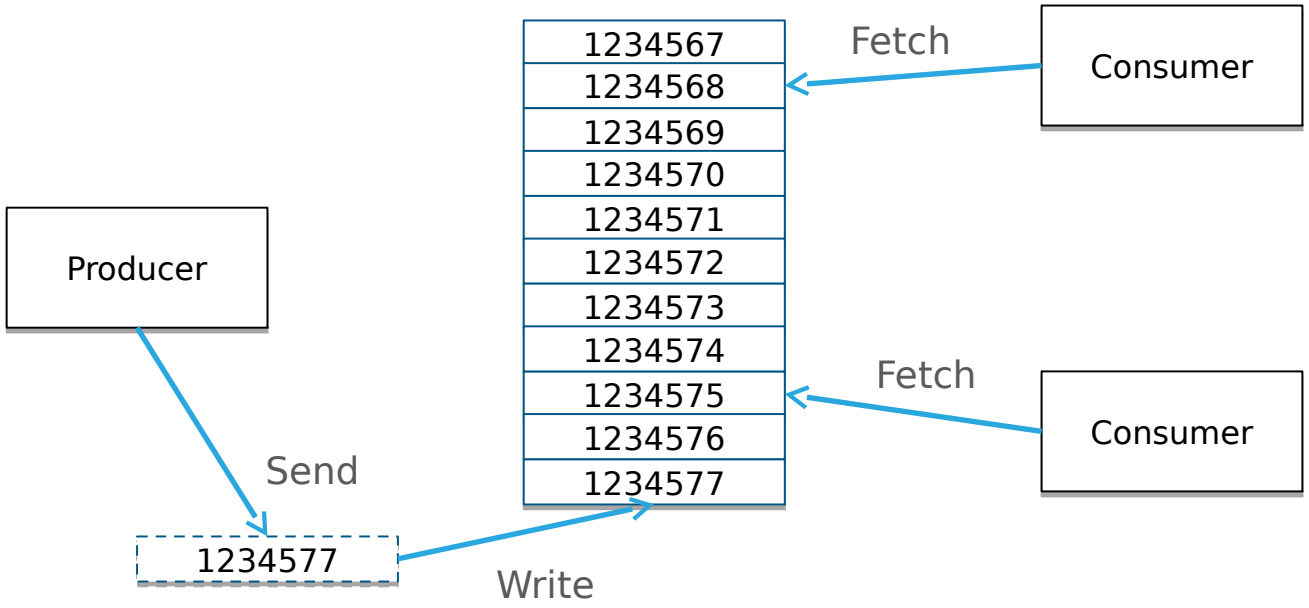
Consumer

- 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



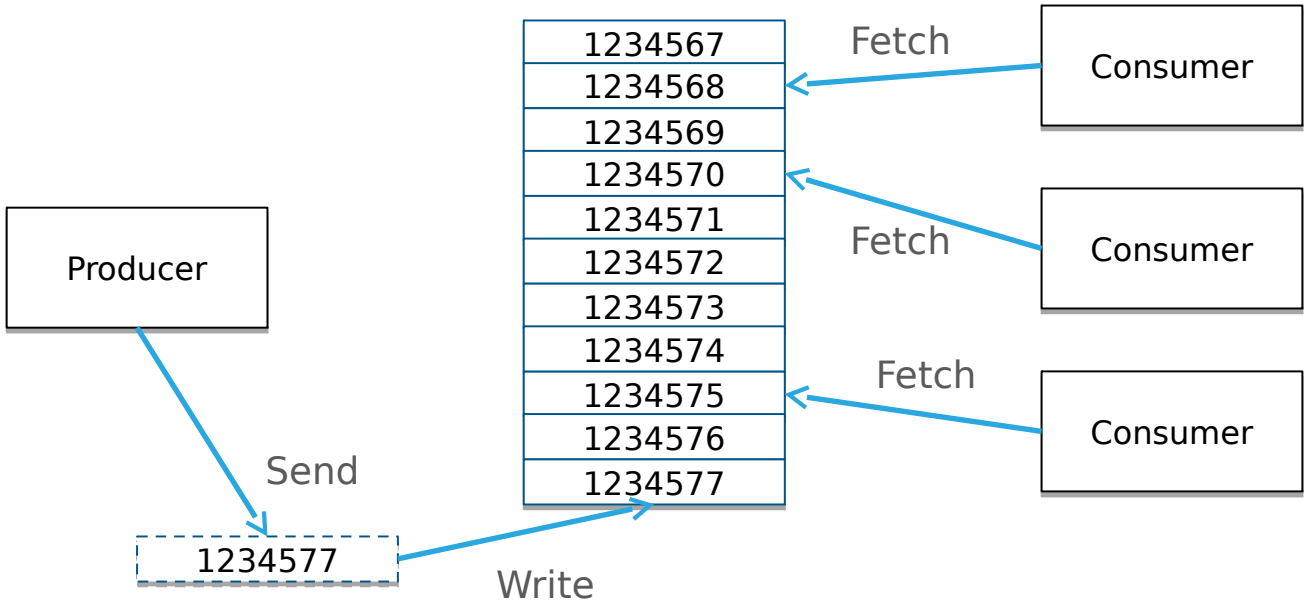
Consumer

- Consumers can go away



Consumer

- And then come back



Consumer - Groups

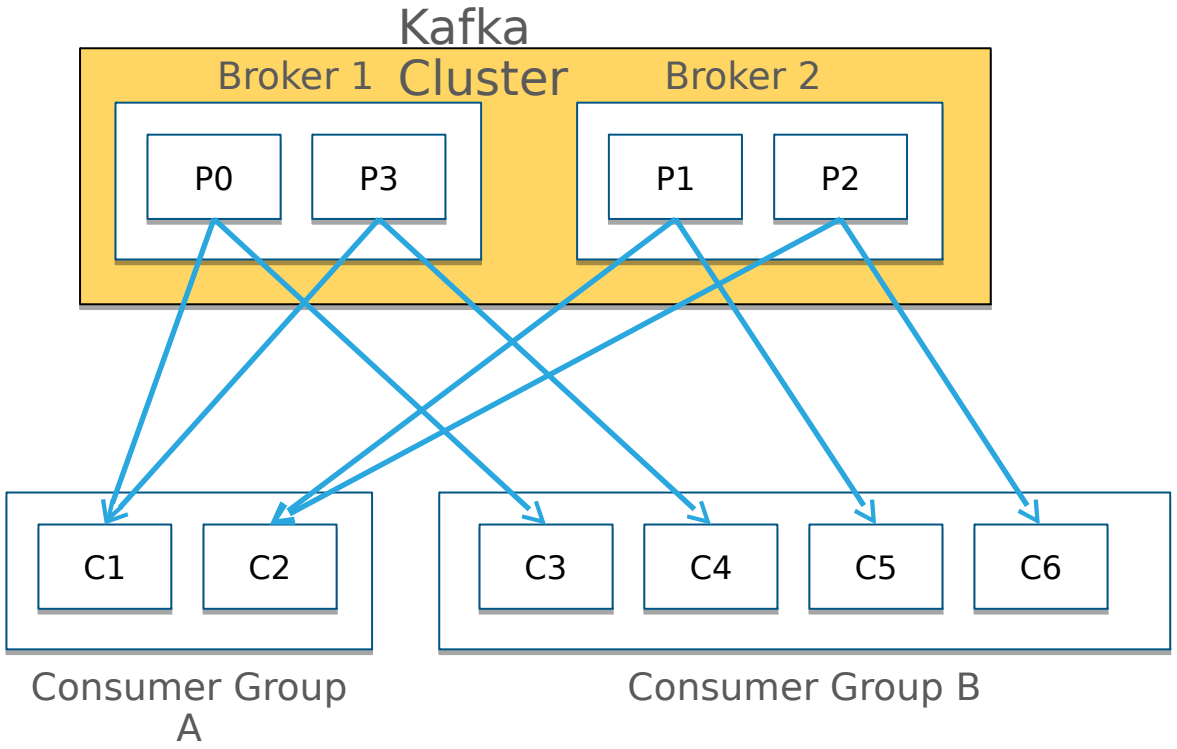
- Consumers can be organized into Consumer Groups

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

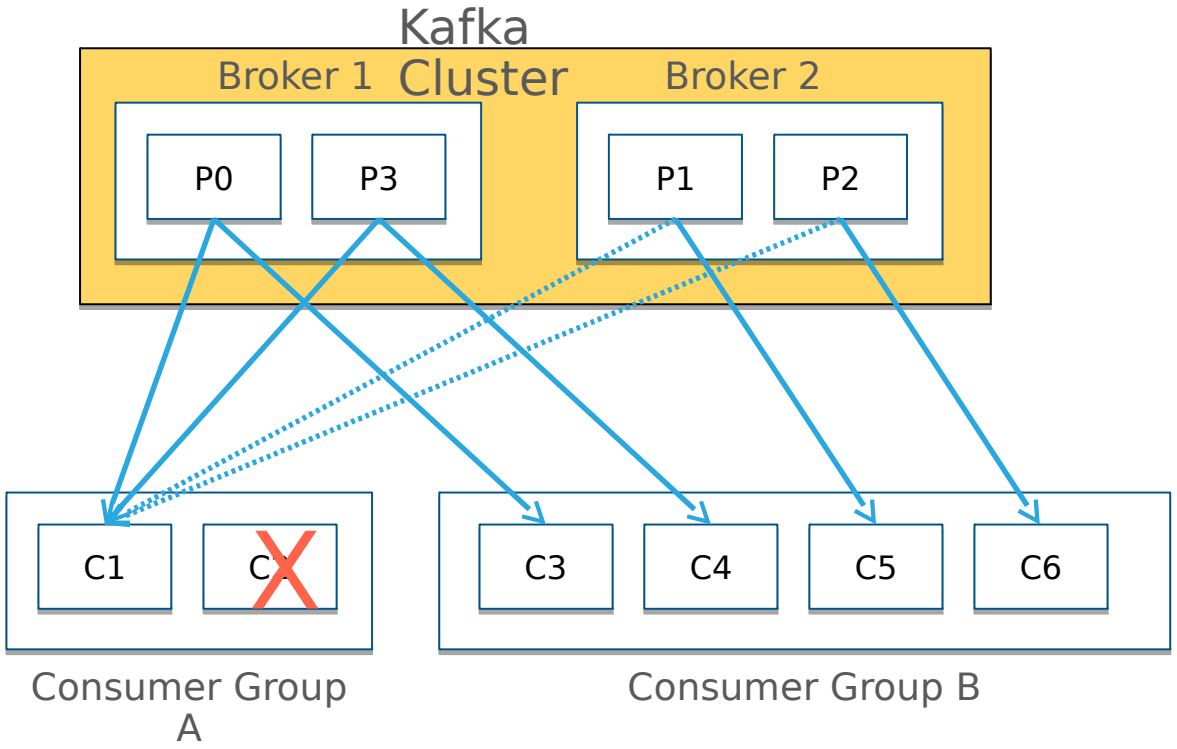
Consumer - Groups

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 ?

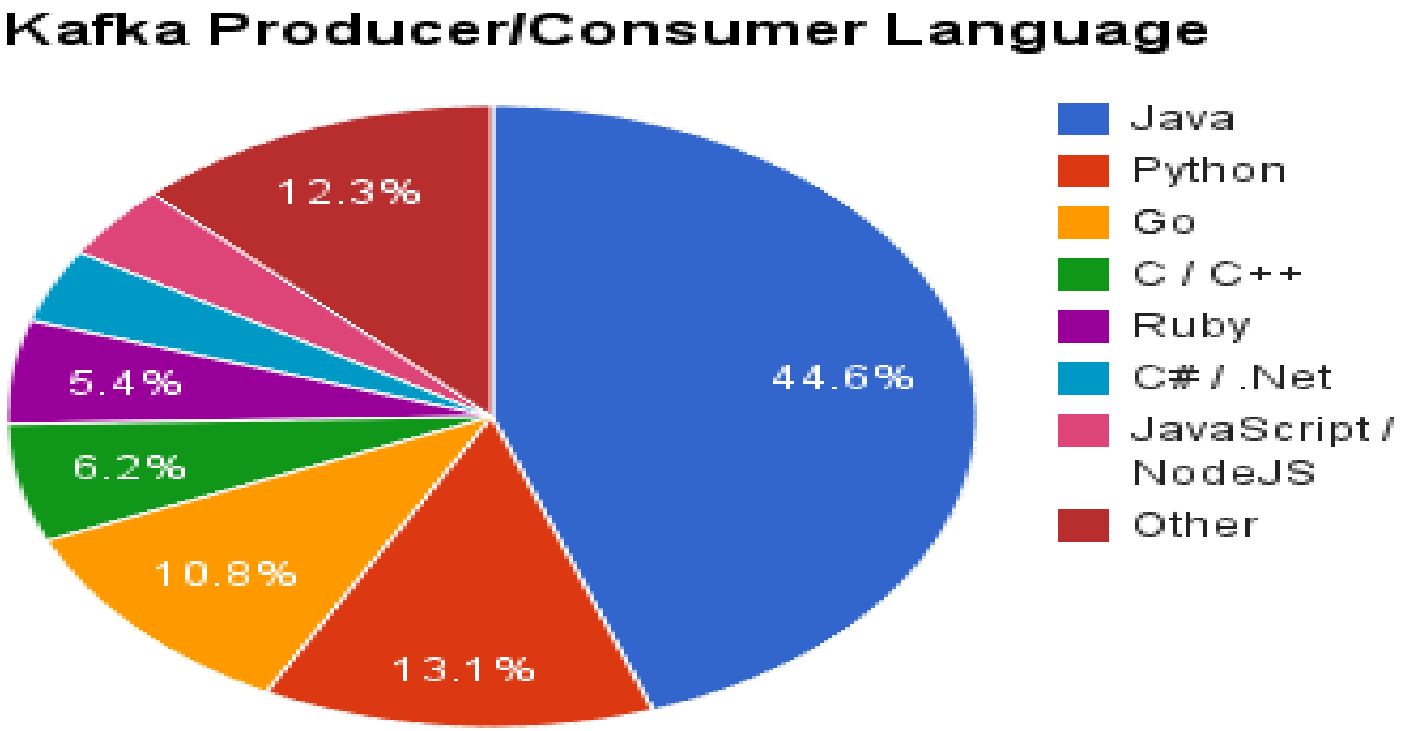
- For really large file transfers?
  - Probably not, it’s designed for “messages” not really for files. If you need to ship large files, consider good-ole-file transfer, or breaking up the files and reading per line to move to Kafka.
- As a replacement for MQ/Rabbit/Tibco
  - Probably. Performance Numbers 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?
  - Kafka-0.9.0.0 onwards kafka provides ssl security and authentication(experimental)
- To do transformations of data?
  - Kafka-0.9.0.0 onwards kafka introduce kafka-connect for this.

Developing with Kafka

API and Clients

Kafka Clients

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



Producer

```
public static void main(String[] args) throws InterruptedException {
    String topic = "kafka";
    String brokers = "localhost:9092";
    String StringSerializer = "org.apache.kafka.common.serialization.StringSerializer";
    Random rnd = new Random();

    Map<String, Object> config = new HashMap<>();
    config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers);
    config.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer);
    config.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer);

    KafkaProducer producer = new KafkaProducer<String, String>(config);

    while (true) {
        LocalDateTime runtime = LocalDateTime.now();
        String ip = "192.168.2." + rnd.nextInt(255);
        String msg = runtime + " www.example.com " + ip;
        System.out.println(msg);
        ProducerRecord record = new ProducerRecord<String, String>(topic, ip, msg);
        producer.send(record);
        Thread.sleep(100);
    }
}
```

Consumer

```
public static void main(String[] args) throws InterruptedException {
    String topic = "kafka";
    String brokers = "localhost:9092";
    String stringSerializer = "org.apache.kafka.common.serialization.StringDeserializer";

    Map<String, Object> config = new HashMap<>();
    config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers);
    config.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, stringSerializer);
    config.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, stringSerializer);
    config.put(ConsumerConfig.GROUP_ID_CONFIG, "test");

    KafkaConsumer consumer = new KafkaConsumer<String, String>(config);
    HashSet<String> topics = new HashSet<>();
    topics.add(topic);
    consumer.subscribe(topics);
    while (true) {
        ConsumerRecords<String, String> records = consumer.poll(200);
        if (!records.isEmpty()) {
            for (ConsumerRecord<String, String> record : records) {
                System.out.println(record);
            }
        }
    }
```

Kafka Clients

- Full list on the wiki, some examples...

Administration

Basic

Installation / Configuration

1. Download kafka from
   - > wget
     http://apache.mirror.serversaustralia.com.au/kafka/0.10.0.0/kafka_2.11-0.10.0.0.tgz
   - > tar -xvzf kafka_2.11-0.10.0.0.tgz
   - > cd kafka_2.11-0.10.0.0
2. Start Zookeeper server
   - > bin/zookeeper-server-start.sh config/zookeeper.properties
3. Start Kafka sever
   - > bin/kafka-server-start.sh config/server.properties

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
```

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

Thank you.