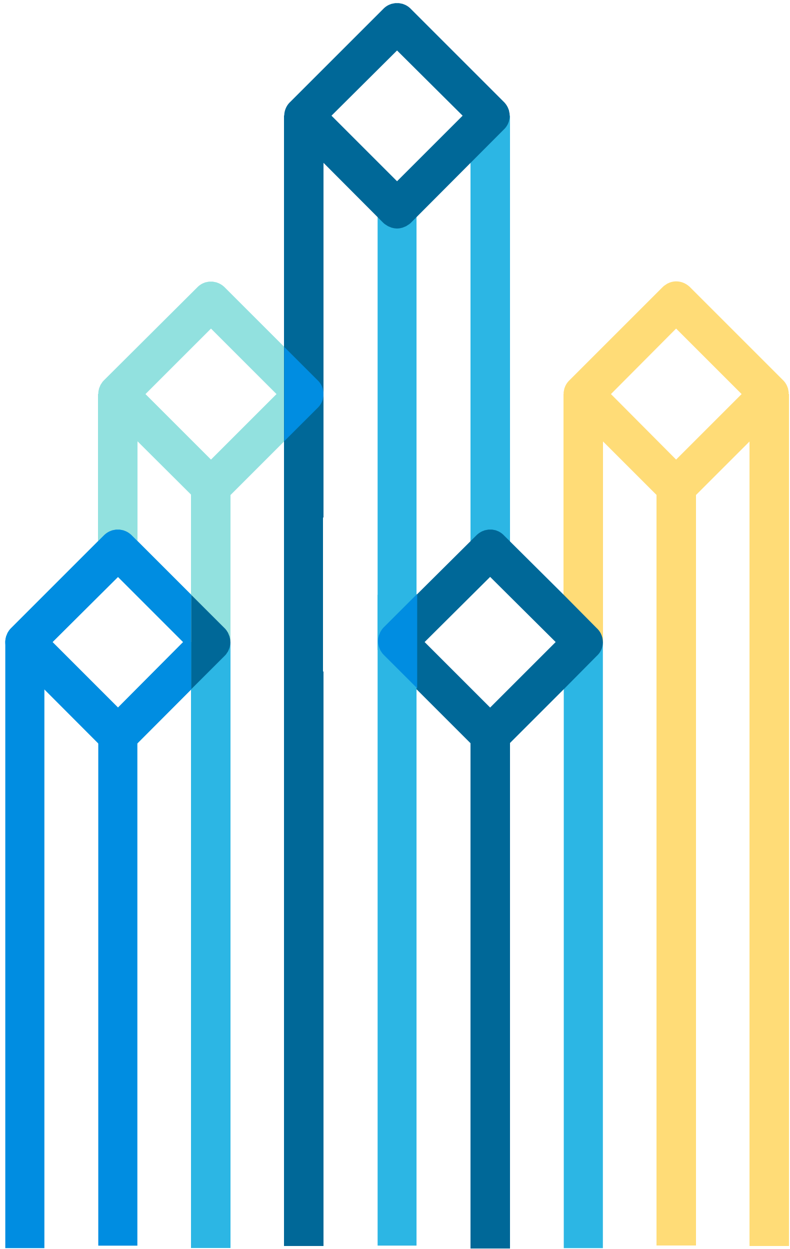
cloudera®

Decoupling Decisions with Apache Kafka

August, 2016



About Me

- Cloudera Kafka Software Engineer
- Distributed Systems Enthusiast
- Father to a 15 month old

grant@cloudera.com

@gchenke

github.com/granthenke

linkedin.com/in/granthenke

cloudera

Agenda

Apache Kafka

Introduction

Terminology

Guarantees

Decoupling Decisions

What?

Getting Started

Command Line

Configurations

Choosing Partitions

cloudera

Apache Kafka

A brief overview

What Is Kafka?

Kafka provides the functionality of a messaging system, but with a unique design.

What Is Kafka?

Kafka is a distributed, partitioned, replicated commit log service.

What Is Kafka?

Kafka is Fast:

A single Kafka broker can handle hundreds of megabytes of reads and writes per second from thousands of clients.

What Is Kafka?

Kafka is Scalable:

Kafka is designed to allow a single cluster to serve as the central data backbone for a large organization.

What Is Kafka?

Kafka is Scalable:

Kafka can be expanded without downtime.

What Is Kafka?

Kafka is Durable:

Messages are persisted and replicated within the cluster to prevent data loss.

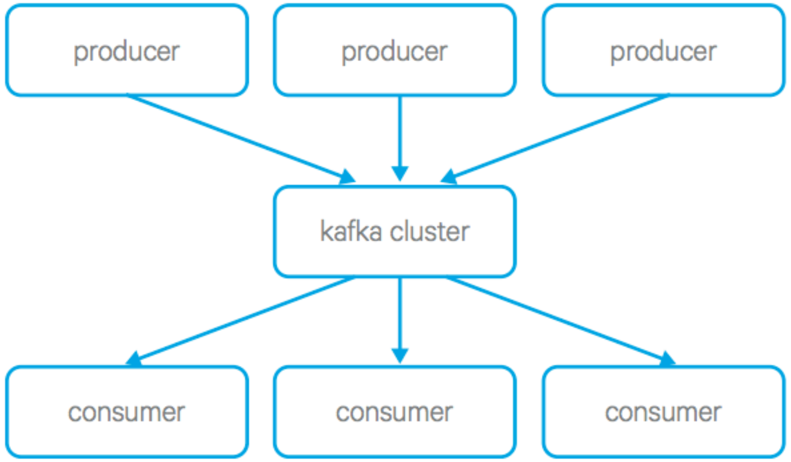
What Is Kafka?

Kafka is Durable:

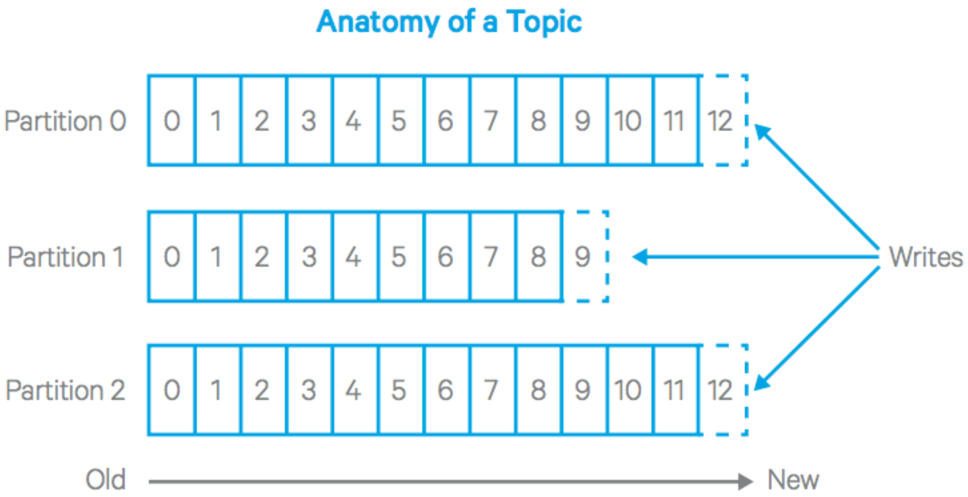
Each broker can handle terabytes of messages without performance impact.

The Basics

- Kafka runs in a cluster. Nodes are called **brokers**
- **Producers** push *messages*
- **Consumers** pull *messages*



- **Messages** are organized into *topics*
- **Topics** are broken into *partitions*
- **Partitions** are replicated across the brokers as **replicas**



Beyond Basics…

Messages

- Optionally be **keyed** in order to map to a static **partition**
  - Used if ordering within a partition is needed
  - Avoid otherwise (extra complexity, skew, etc.)
- Location of a message is denoted by its topic, partition & **offset**
  - A partitions offset increases as messages are appended

Replicas

- A partition has 1 leader replica. The others are followers.
- Followers are considered in-sync when:
  - The replica is alive
  - 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)
- Replicas map to physical locations on a broker

Kafka Guarantees

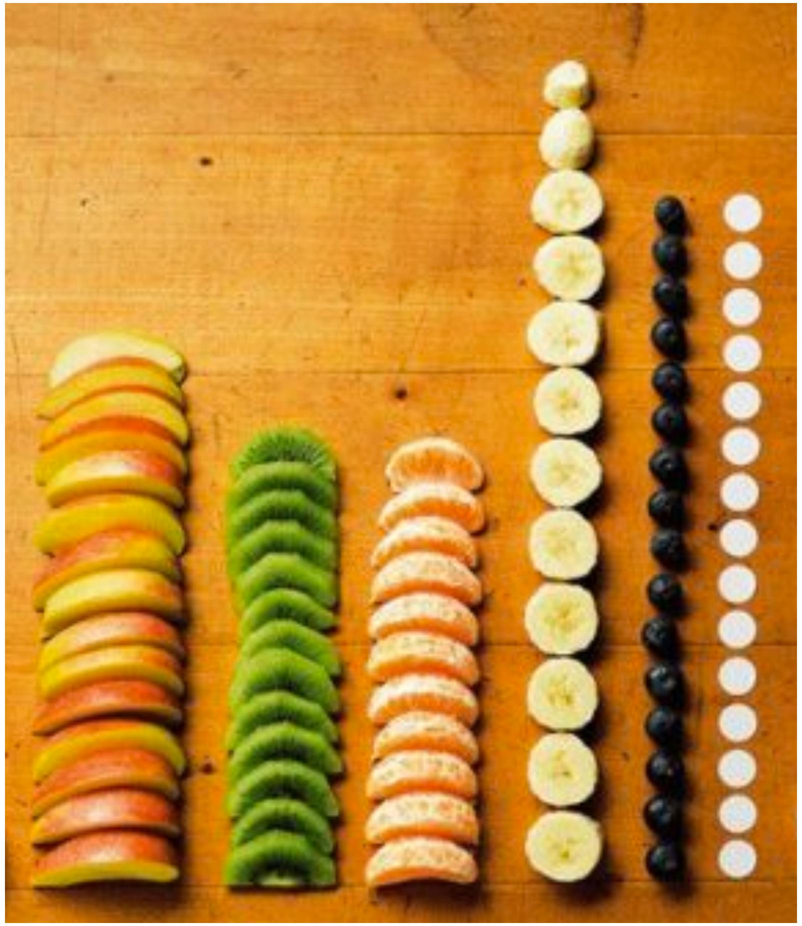
Kafka Guarantees

WARNING: Guarantees can vary based on your configuration choices.

Kafka Guarantees: Message Ordering

- Messages sent to each partition will be appended to the log in the order they are sent

- Messages read from each partition will be seen in the order stored in the log



cloudera

Kafka Guarantees: Message Delivery

- At-least-once: Messages are never lost but may be redelivered
- Duplicates can be minimized but not totally eliminated
- Generally only get duplicates during failure or rebalance scenarios
- It’s a good practice to build pipelines with duplicates in mind regardless

Kafka Guarantees: Message Safety

- Messages written to Kafka are durable and safe

- Once a published message is committed it will not be lost as long as one broker that replicates the partition to which this message was written remains "alive”

- Only committed messages are ever given out to the consumer. This means that the consumer need not worry about potentially seeing a message that could be lost if the leader fails.

cloudera®

Decoupling Decisions

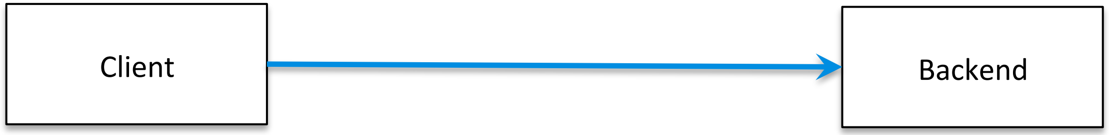
Flexible from the beginning

How It Starts

- Data pipelines start simple
- One or two data sources
- One backend application

Initial Decisions:

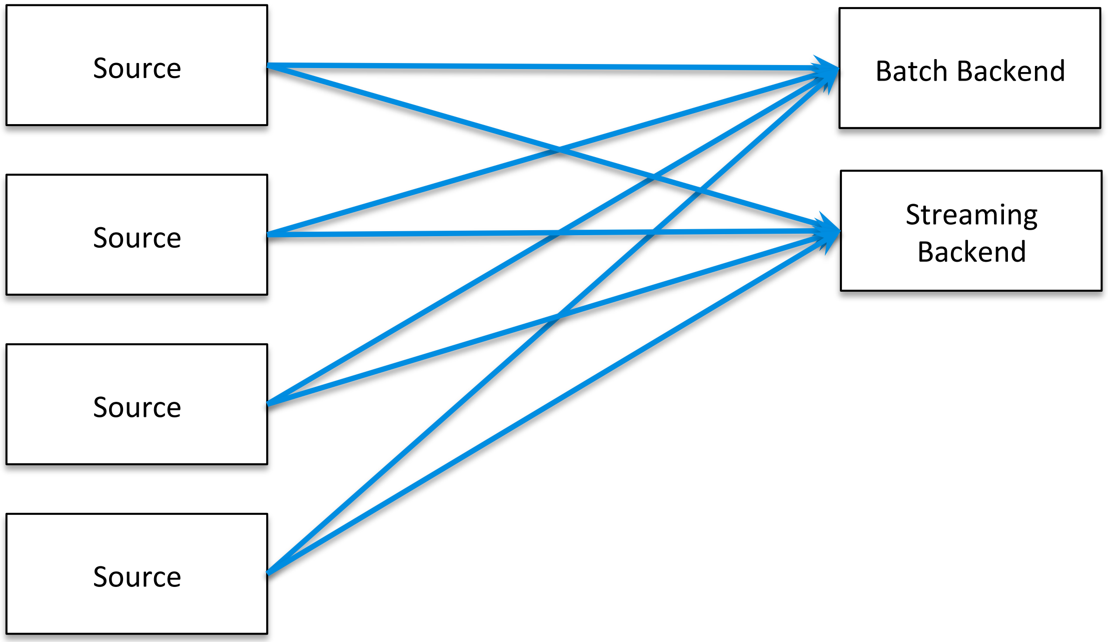
- How can I be successful quickly?
- What does this specific pipeline need?
- Don’t prematurely optimize



cloudera

Then Quickly…

- Multiple sources
- Another backend application
- Initial decisions need to change

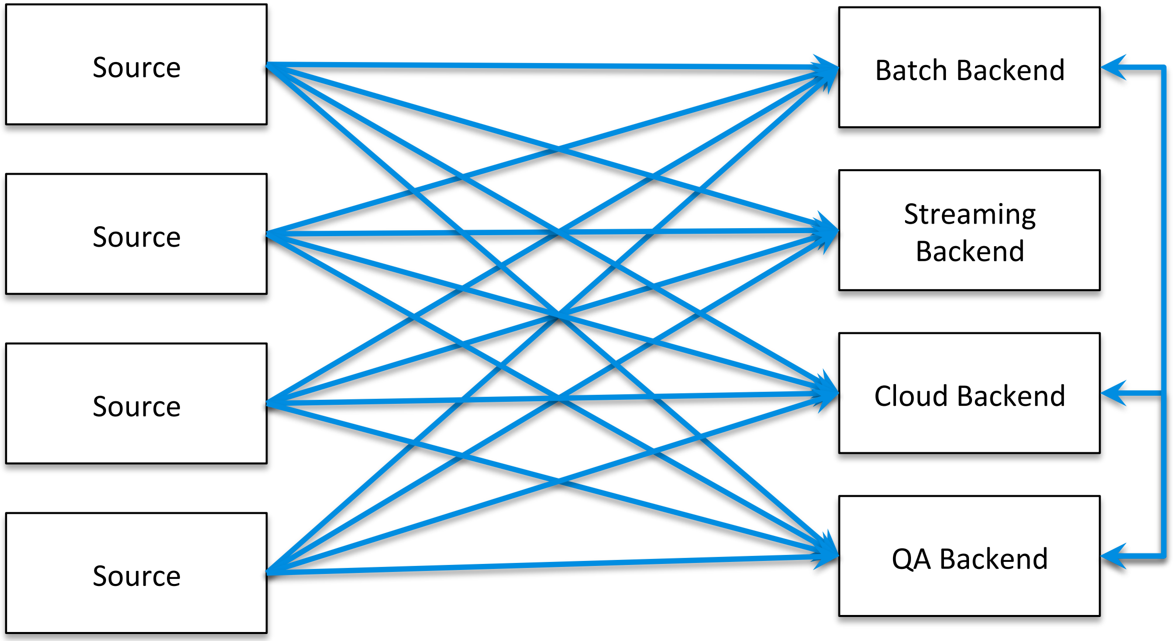


And Eventually…

- More environments

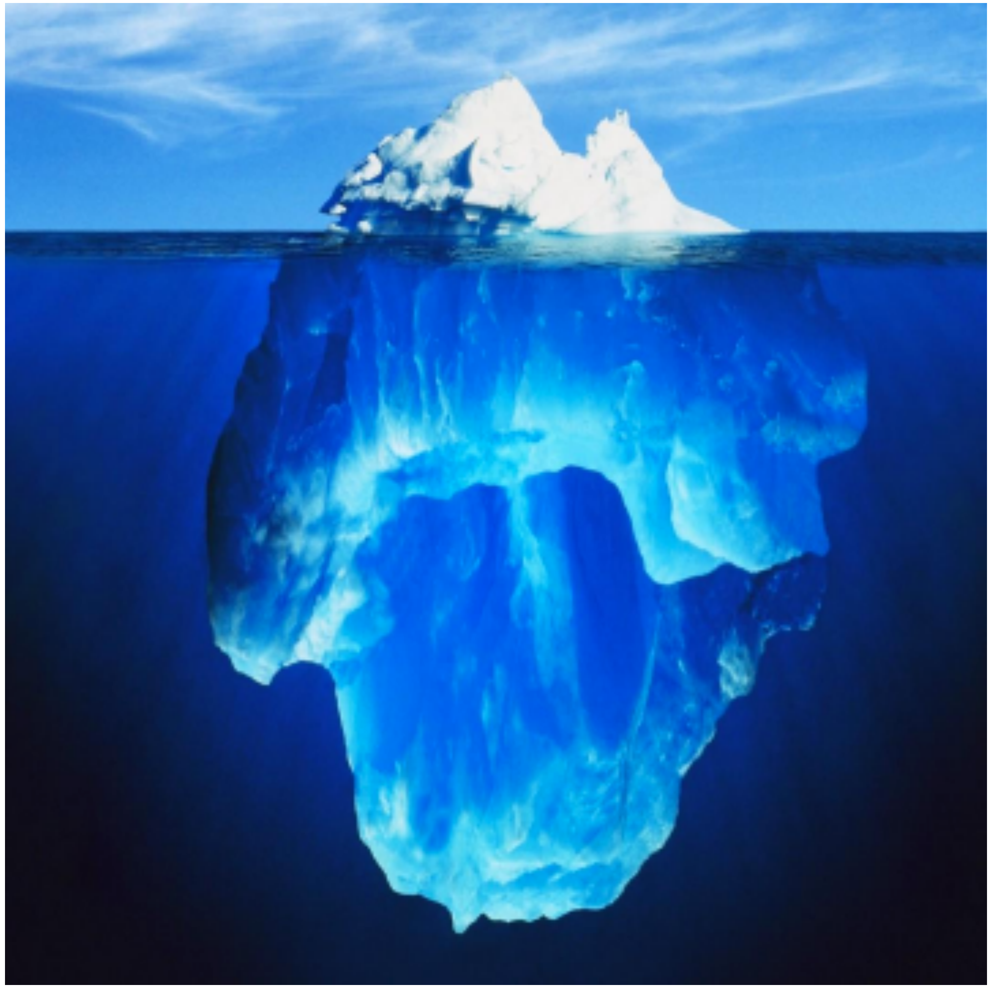
- Backend applications feed other backend applications

- You may also want to
  - Experiment with new software
  - Change data formats
  - Move to a streaming architecture



Technical Debt

- Early decisions made for that single pipeline have impacted each system added
- Because sources and applications are tightly coupled change is difficult
- Progress becomes slower and slower
- The system has grown fragile
- Experimentation, growth, and innovation is risky



Decision Types: Type 1 decisions

“Some decisions are consequential and irreversible or nearly irreversible – one-way doors – and these decisions must be made methodically, carefully, slowly, with great deliberation and consultation…” —Jeff Bezos

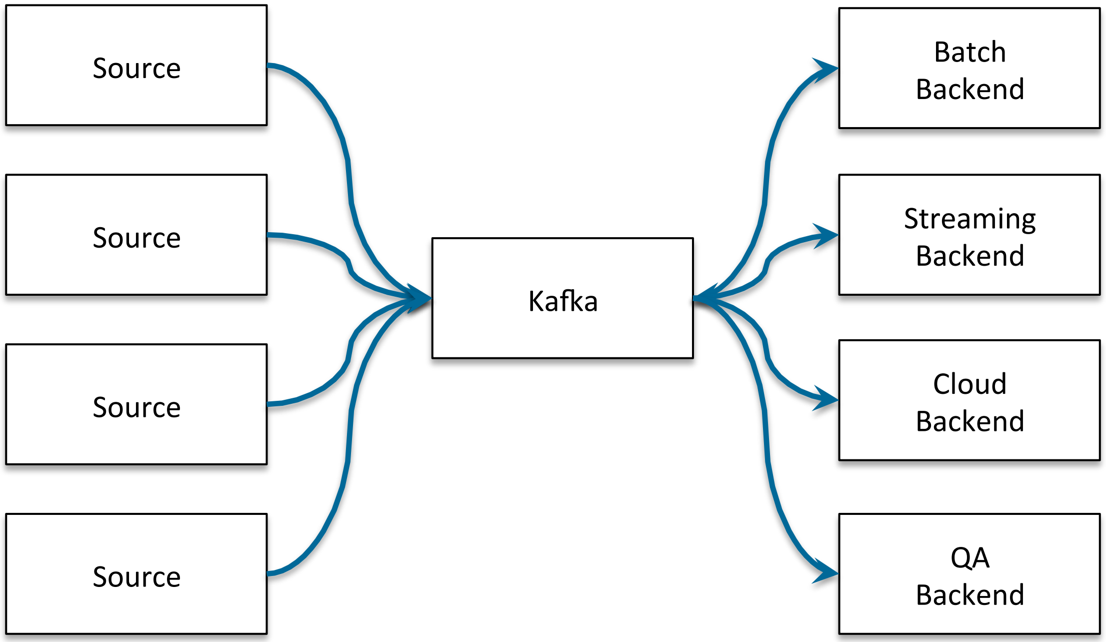
Decision Types: Type 2 Decisions

"Type 2 decisions are changeable, reversible – they're two-way doors. If you've made a suboptimal Type 2 decision, you don't have to live with the consequences for that long."—Jeff Bezos

Kafka Is Here To Help!

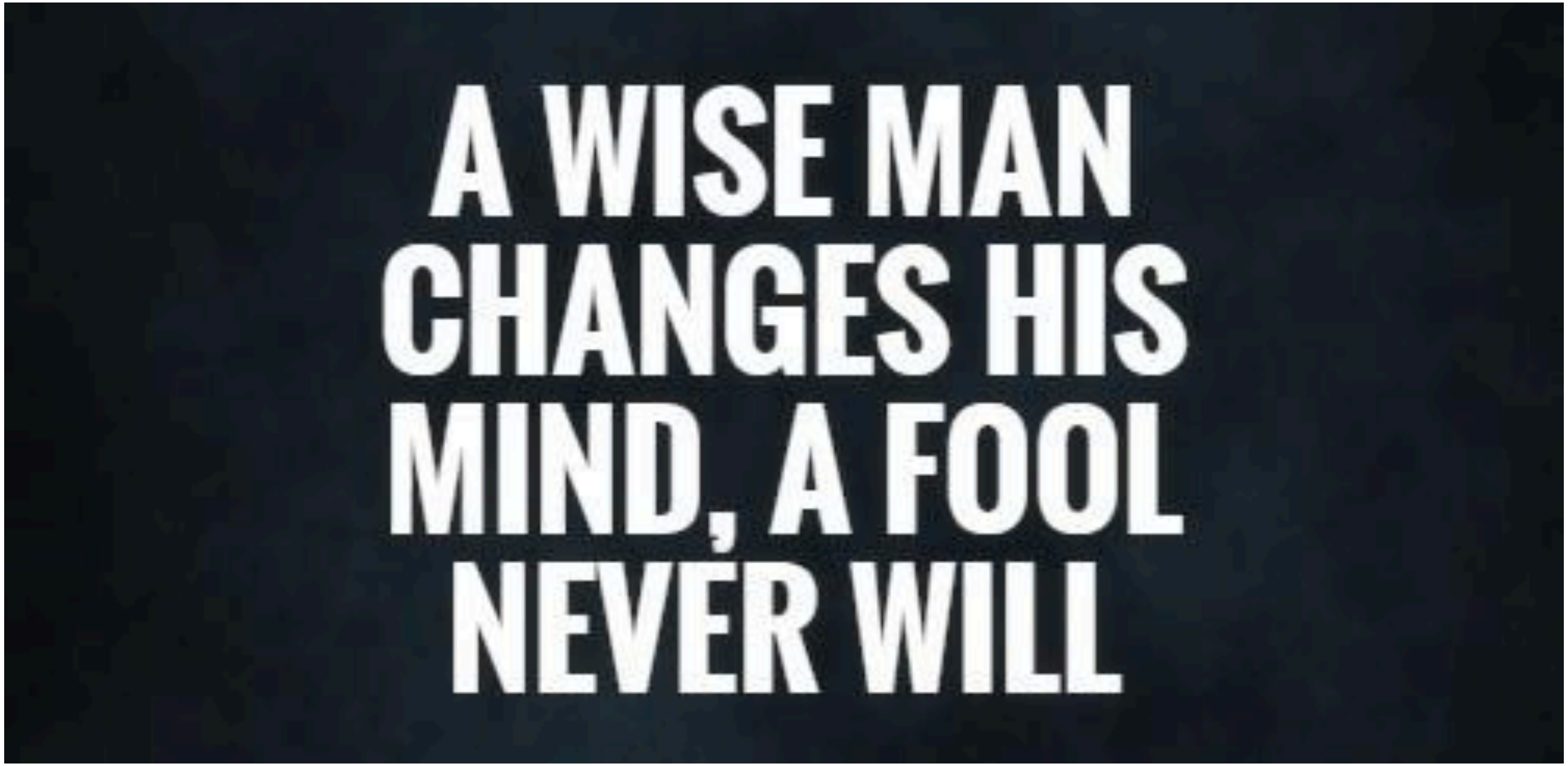
With Kafka

- A central backbone for the entire system
- Decouples source and backend systems
  - Slow or failing consumers don't impact source system
- Adding new sources or consumers is easy and low impact



cloudera

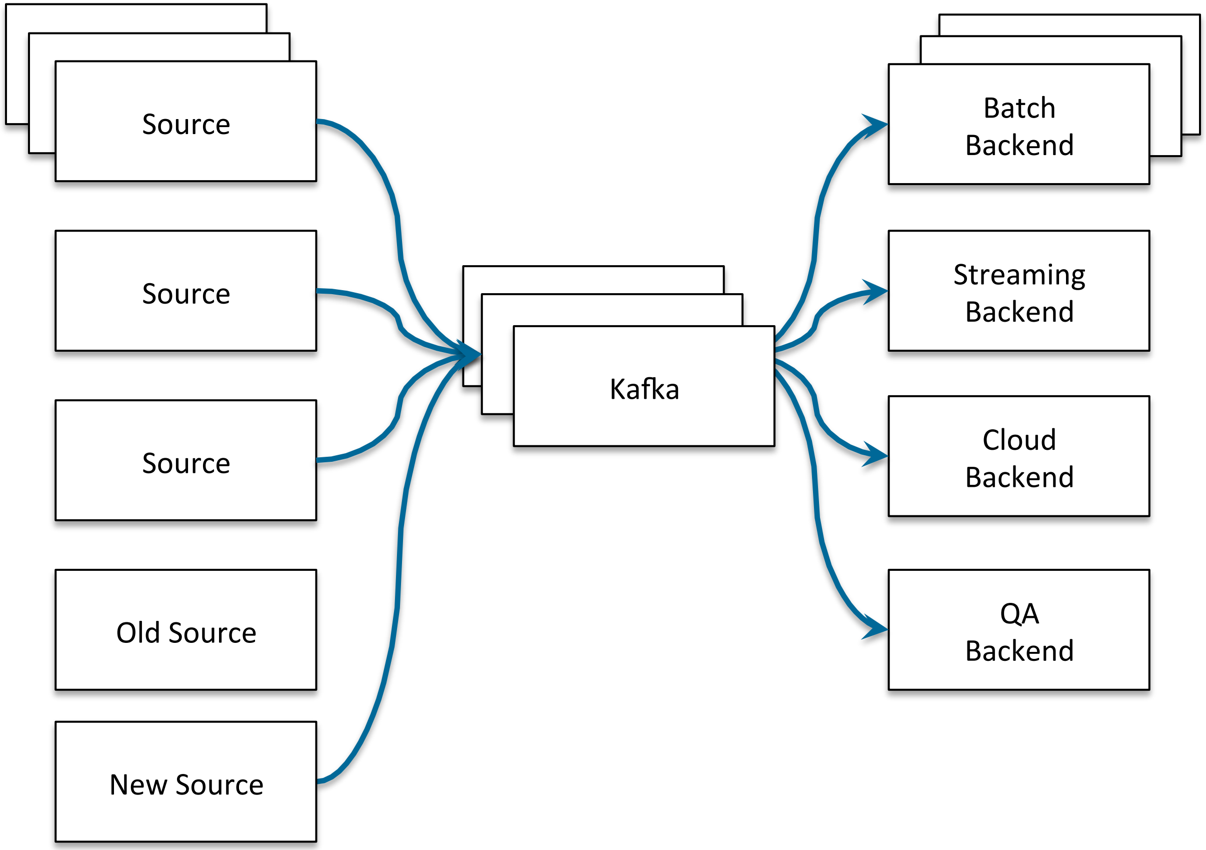
Lets Make Some Changes



A WISE MAN CHANGES HIS MIND, A FOOL NEVER WILL

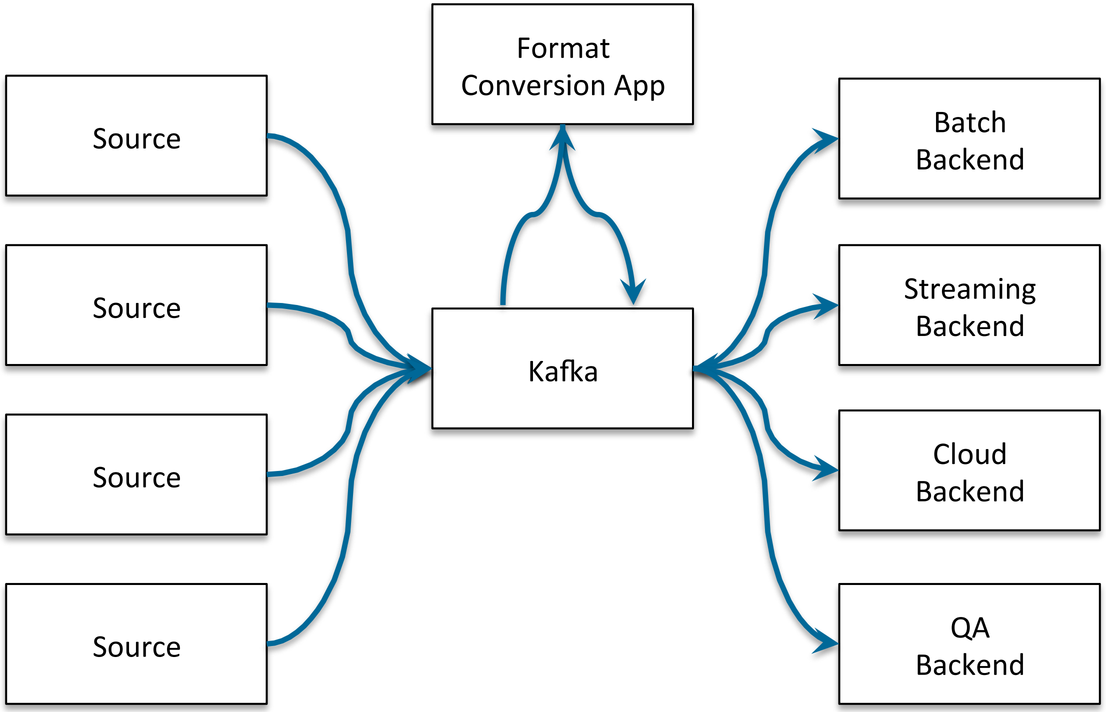
The Really Easy Changes

- Add new source or backend
- Process more data
- Move from batch to streaming
- Change data source



Change Data Format

- I would like to support avro (or thrift, protobuf, xml, json, ...)
- Keep source data raw
- In a streaming application transform formats
- Read from source-topic and produce to source-topic-{format}
- This could also include lossy/ optimization transformations



cloudera

Change Business Logic

- Deploy new application and replay the stream

- Great for testing and development

- Extremely useful for handling failures and recovery too

Change Application Language

- There are well written clients in a lot of programming languages

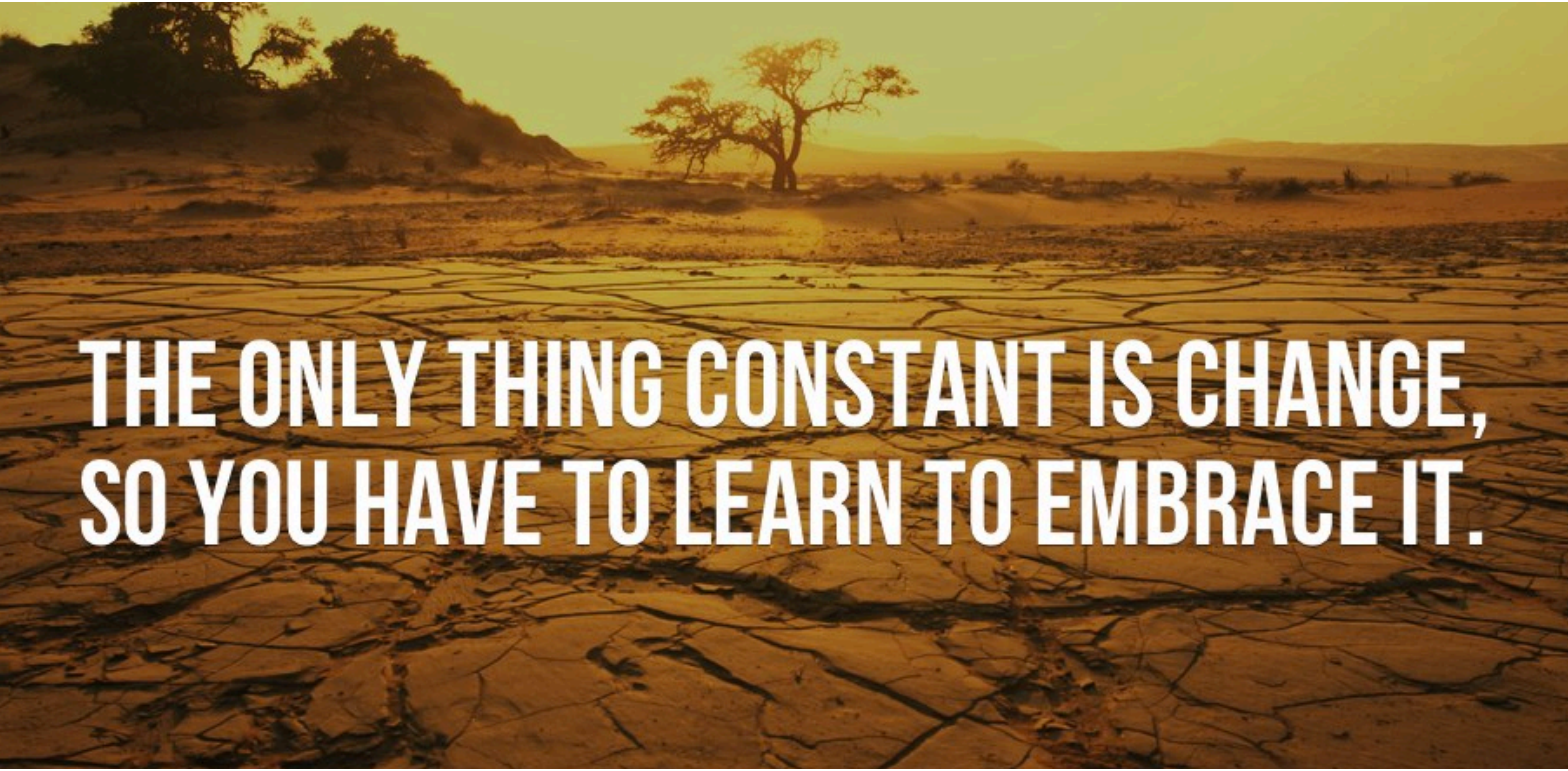
- In the rare case your language of choice doesn't have a client, you can use the binary wire protocol and write one

Change Processing Framework

- Many processing frameworks get Kafka integration early on

- Because consumers don’t affect source applications its safe to experiment

Spark
kafka Streams
FLUME
STORM
beam
Flink



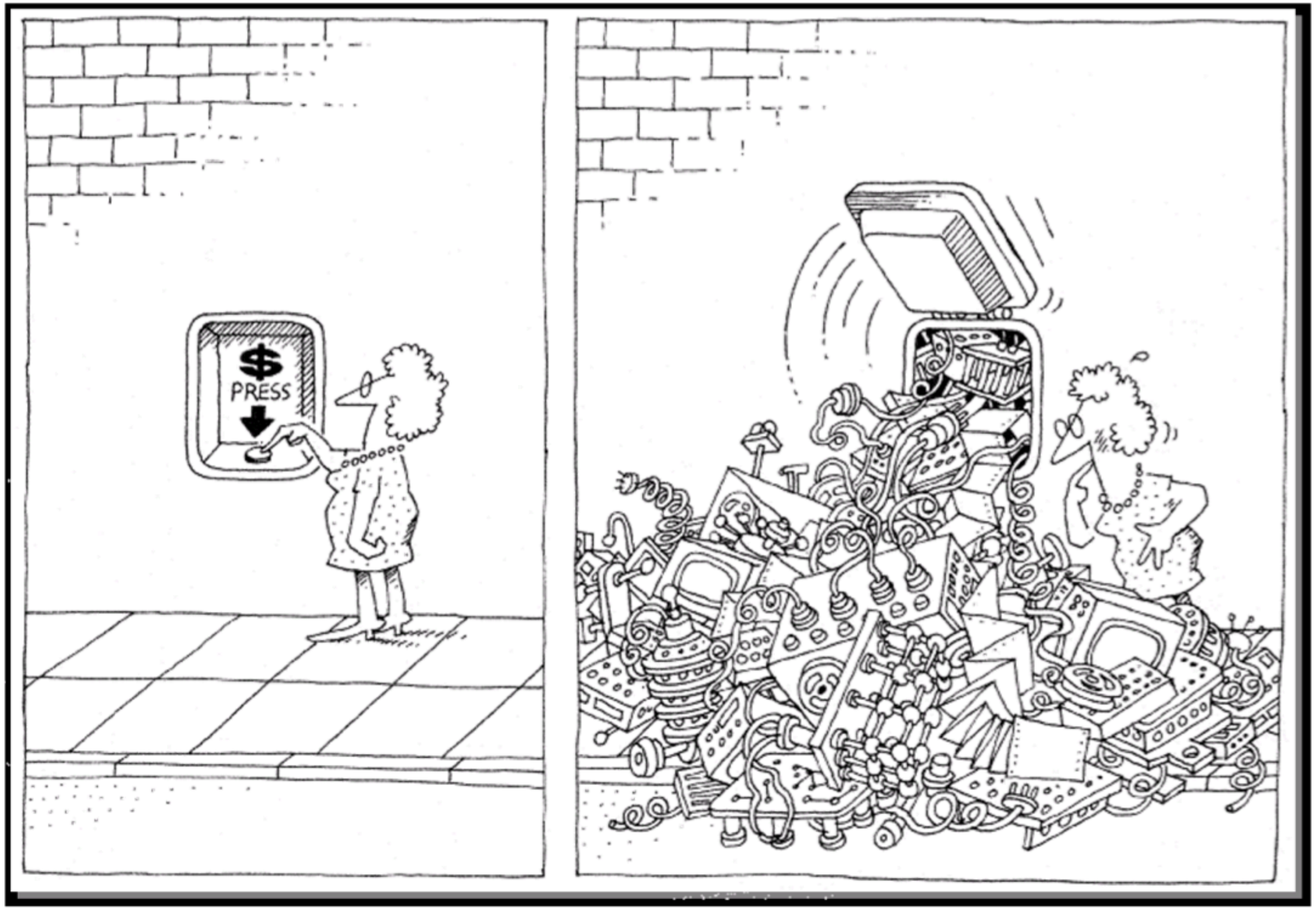
THE ONLY THING CONSTANT IS CHANGE, SO YOU HAVE TO LEARN TO EMBRACE IT.

cloudera®

Quick Start

Sounds great...but how do I use it?

Let’s Keep it Simple

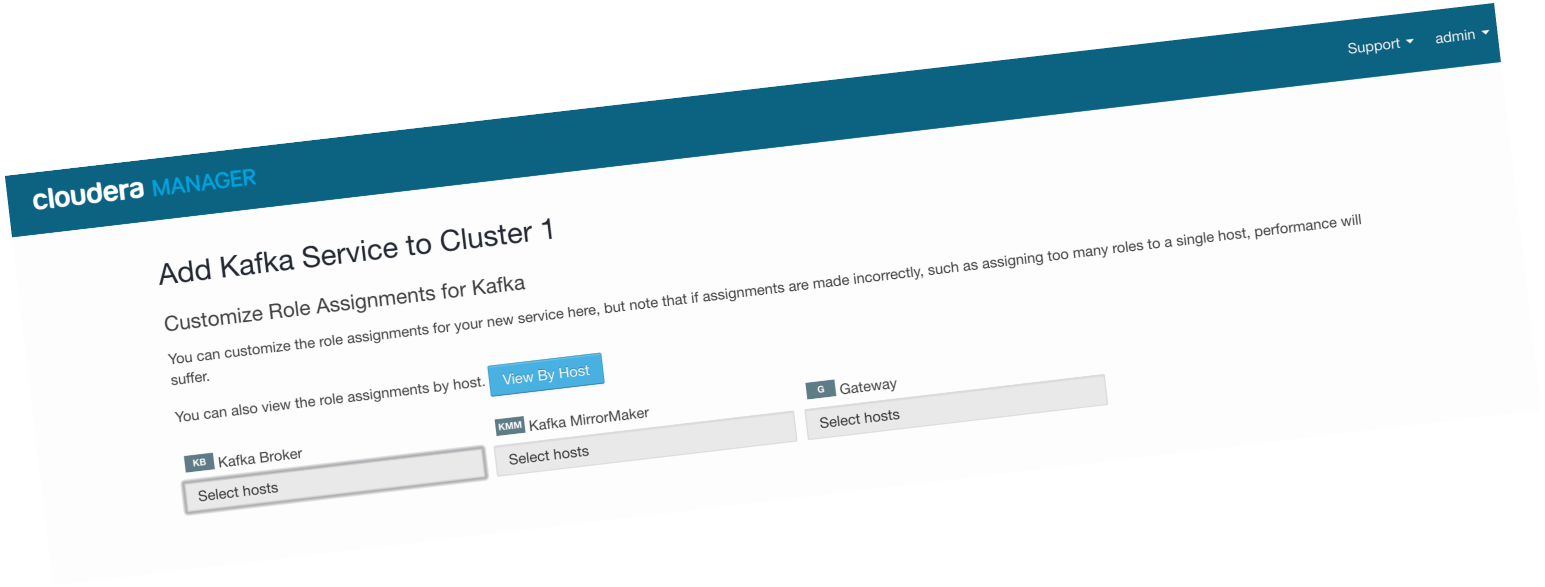


Install Kafka

```
> wget http://apache.claz.org/kafka/0.10.0.0/kafka_2.11-0.10.0.0.tgz
> tar -xzf kafka_2.11-0.10.0.0.tgz
> cd kafka_2.11-0.10.0.0
```

```
> brew install kafka
```

```
$ sudo yum clean all
$ sudo yum install kafka
$ sudo yum install kafka-server
```



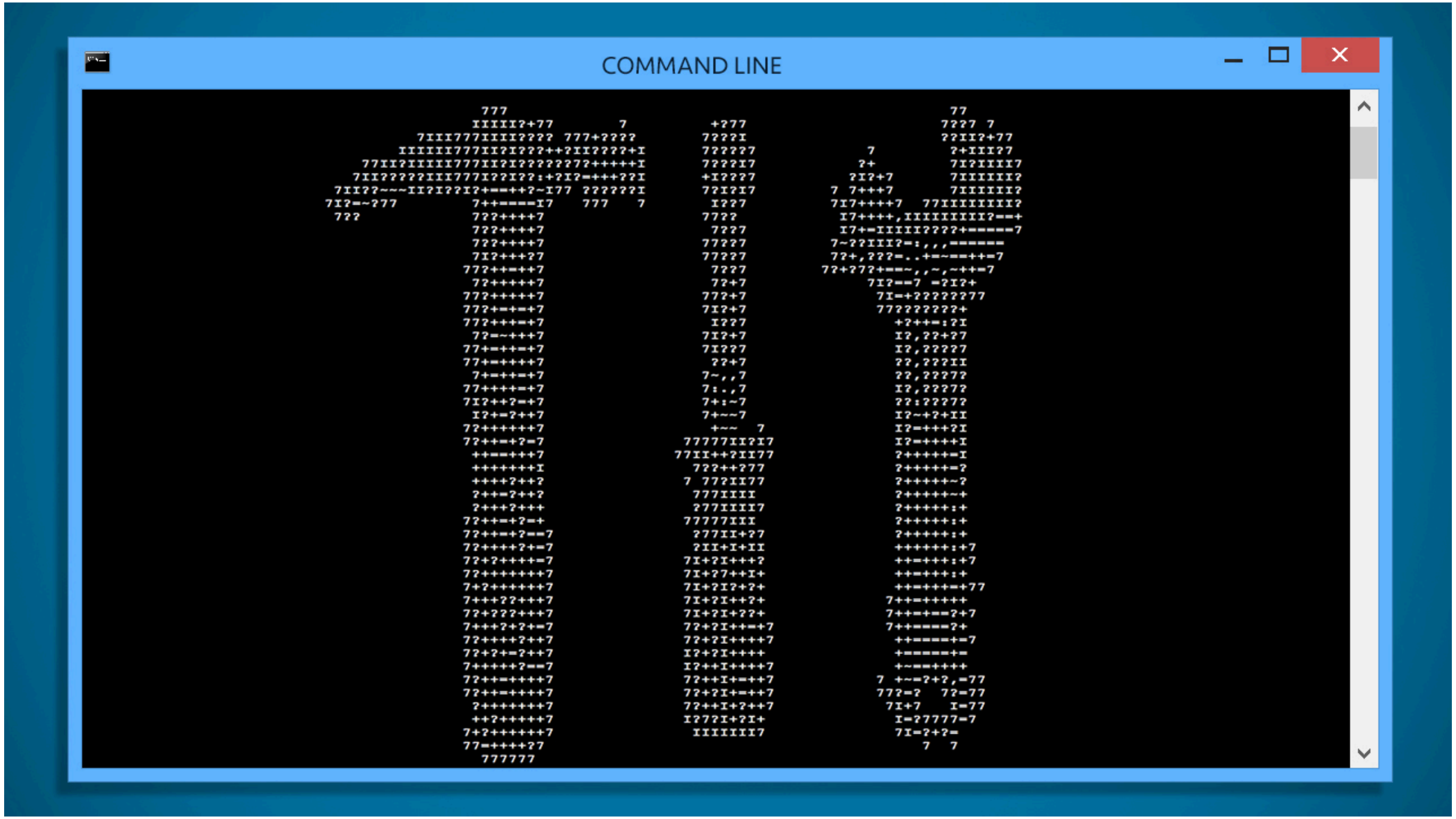
Add Kafka Service to Cluster 1

Customize Role Assignments for Kafka

You can customize the role assignments for your new service here, but note that if assignments are made incorrectly, such as assigning too many roles to a single host, performance will suffer.

You can also view the role assignments by host.

Start with the CLI tools



```
# Create a topic & describe
kafka-topics --zookeeper my-zk-host:2181 --create --topic my-topic --partitions 10
  --replication-factor 3
kafka-topics --zookeeper my-zk-host:2181 --describe --topic my-topic

# Produce in one shell
vmstat -w -n -t 1 | kafka-console-producer --broker-list my-broker-host:9092 --
  topic my-topic

# Consume in a separate shell
kafka-console-consumer --zookeeper my-zk-host:2181 --topic my-topic
```

```
# Create a topic & describe
kafka-topics --zookeeper my-zk-host:2181 --create --topic my-topic --partitions 10 --replication-factor 3
kafka-topics --zookeeper my-zk-host:2181 --describe --topic my-topic
```

```
# Produce in one shell
vmstat -w -n -t 1 | kafka-console-producer --broker-list my-broker-host:9092 --topic my-topic
```

```
# Consume in a separate shell
kafka-console-consumer --zookeeper my-zk-host:2181 --topic my-topic
```

```
# Create a topic & describe
kafka-topics --zookeeper my-zk-host:2181 --create --topic my-topic --partitions 10 --replication-factor 3
kafka-topics --zookeeper my-zk-host:2181 --describe --topic my-topic

# Produce in one shell
vmstat -w -n -t 1 | kafka-console-producer --broker-list my-broker-host:9092 --topic my-topic

# Consume in a separate shell
kafka-console-consumer --zookeeper my-zk-host:2181 --topic my-topic
```

```
# Create a topic & describe
kafka-topics --zookeeper my-zk-host:2181 --create --topic my-topic --partitions 10
  --replication-factor 3
kafka-topics --zookeeper my-zk-host:2181 --describe --topic my-topic

# Produce in one shell
vmstat -w -n -t 1 | kafka-console-producer --broker-list my-broker-host:9092 --
  topic my-topic

# Consume in a separate shell
kafka-console-consumer --zookeeper my-zk-host:2181 --topic my-topic
```

Kafka Configuration

A starting point

Flexible Configuration

- Tune for throughput or safety
- At least once or at most once
- Per topic overrides and client overrides

Broker Configuration

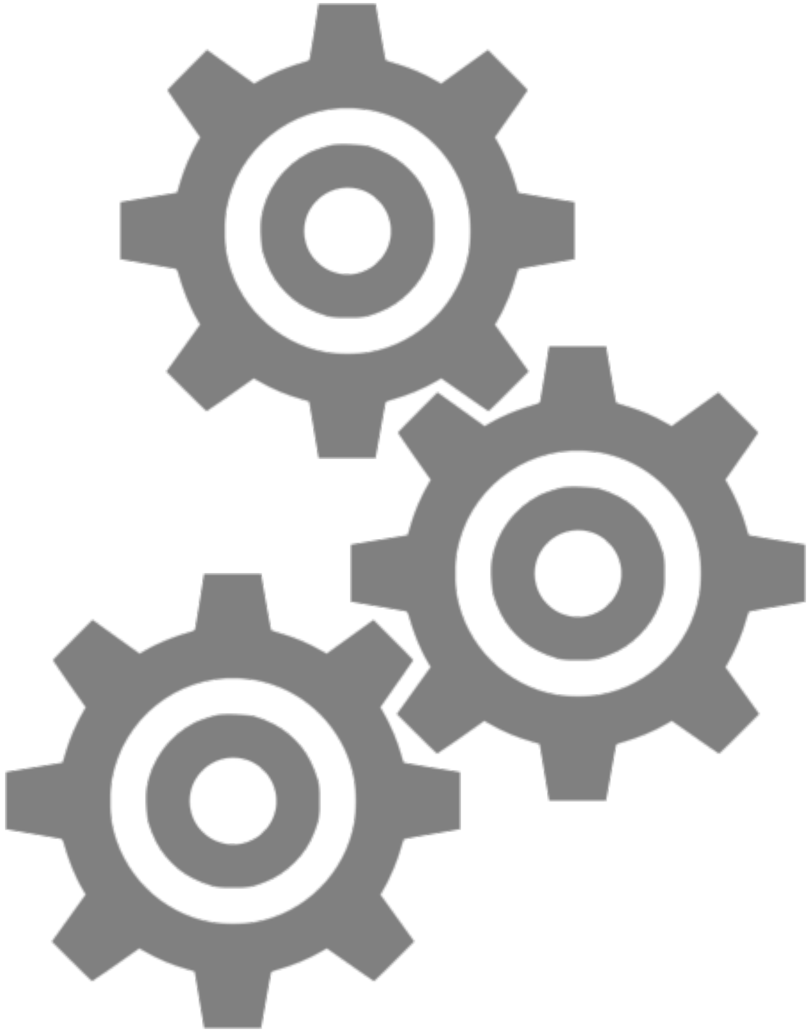
- 3 or more Brokers
- broker_max_heap_size=8GiB
- zookeeper.chroot=kafka
- auto.create.topics.enable=false
  - If you must use it make sure you set
    - num.partitions >= #OfBrokers
    - default.replication.factor=3
- min.insync.replicas=2
- unclean.leader.election=false (default)

Producer Configuration

- Use the new Java Producer
- acks=all
- retries=Integer.MAX_VALUE
- max.block.ms=Long.MAX_VALUE
- max.in.flight.requests.per.connection=1
- linger.ms=1000
- compression.type=snappy

Consumer Configuration

- Use the new Java Consumer
- group.id represents the “Coordinated Application”
  - Consumers within the group share the load
- auto.offset.reset = latest/earliest/none
- enable.auto.commit=false



cloudera

Choosing Partition Counts: Quick Pick

- Just getting started, don’t overthink it
- Don’t make the mistake of picking (1 partition)
- Don’t pick way too many (1000 partitions)
- Often a handwave choice of 25 to 100 partitions is a good start
- Tune when you can understand your data and use case better

What’s Next?

Make something

Getting started is the hardest part

cloudera

Thank you

Common Questions

How do I size broker hardware?

Brokers

- Similar profile to data nodes
- Depends on what's important
  - Message Retention = Disk Size
  - Client Throughput = Network Capacity
  - Producer Throughput = Disk I/O
  - Consumer Throughput = Memory

cloudera

Kafka Cardinality—What is large?

- Brokers: 3->15 per Cluster
  - Common to start with 3-5
  - Very large are around 30-40 nodes
  - Having many clusters is common

- Topics: 1->100s per Cluster

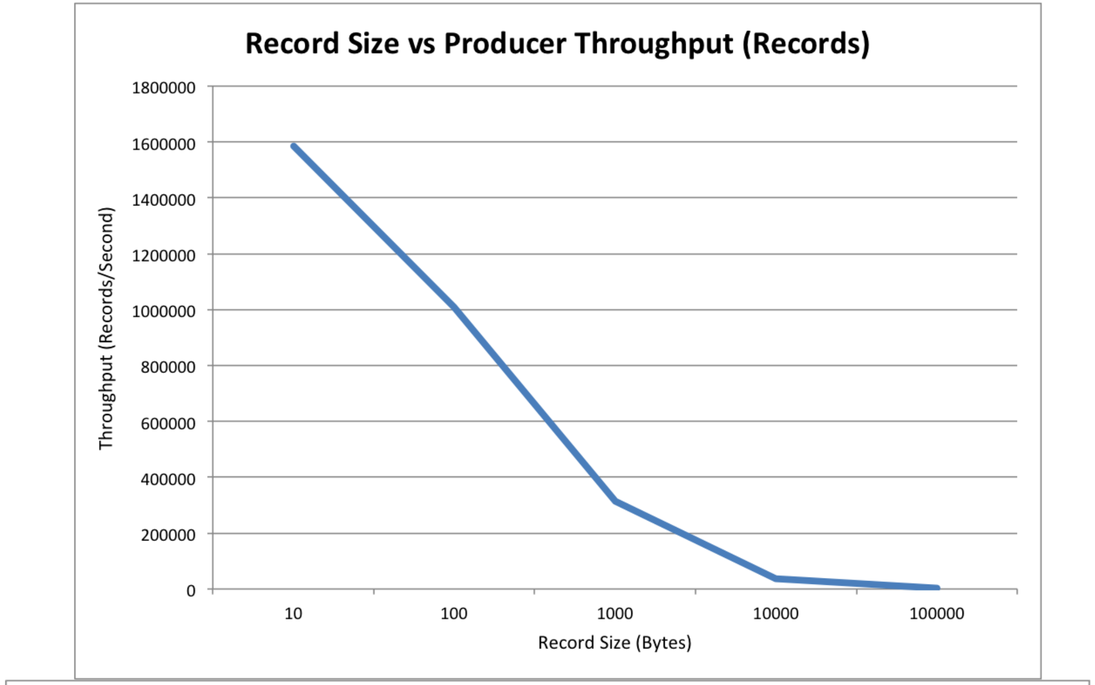
- Partitions: 1->1000s per Topic
  - Clusters with up to 10k total partitions are workable. Beyond that we don't aggressively test. [src]

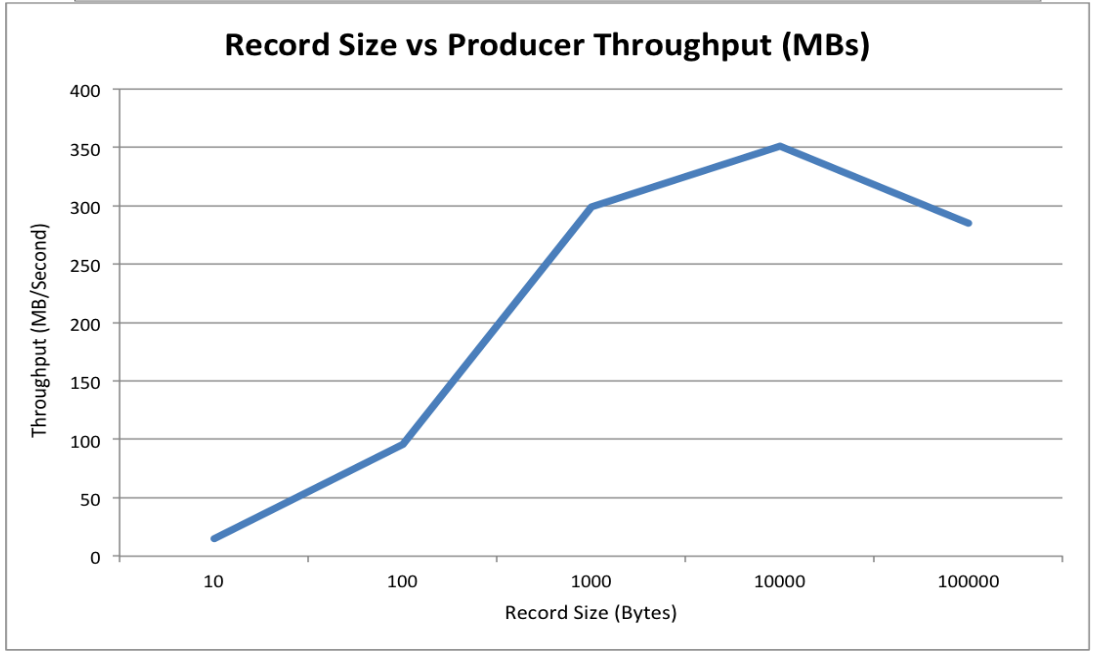
- Consumer Groups: 1->100s active per Cluster
  - Could Consume 1 to all topics

cloudera

Large Messages

- Kafka is not designed for very large messages
- Optimal performance ~10KB
- Could consider breaking up the messages/files into smaller chunks





cloudera

Should I use Raid 10 or JBOD?

RAID10

- Can survive single disk failure
- Single log directory
- Lower total I/O

JBOD

- Single disk failure kills broker
- More available disk space
- Higher write throughput
- Broker is not smart about balancing partitions across disk

Do I need a separate Zookeeper for Kafka?

- It's not required but preferred

- Kafka relies on Zookeeper for cluster metadata & state

- Correct Zookeeper configuration is most important

cloudera

Zookeeper Configuration

- ZooKeeper's transaction log must be on a dedicated device (A dedicated partition is not enough) for optimal performance
  - ZooKeeper writes the log sequentially, without seeking
  - Set **dataLogDir** to point to a directory on that device
  - Make sure to point **dataDir** to a directory not residing on that device

- Do not put ZooKeeper in a situation that can cause a swap
  - Therefore, make certain that the maximum heap size given to ZooKeeper is not bigger than the amount of real memory available to ZooKeeper

Choosing Partition Counts

- A topic partition is the unit of parallelism in Kafka
- It is easier to increase partitions than it is reduce them
  - Especially when using keyed messages
- Consumers are assigned partitions to consume
  - They can’t split/share partitions
  - Parallelism is bounded by the number of partitions

Choosing Partition Counts: Quick Pick

- Just getting started, don’t overthink it
- Don’t make the mistake of picking (1 partition)
- Don’t pick way too many (1000 partitions)
- Often a handwave choice of 25 to 100 partitions is a good start
- Tune when you can understand your data and use case better

Choosing Partition Counts: Estimation

Given:

- pt = production throughput per partition
- ct = consumption throughput per partition
- tt = total throughput you want to achieve
- pc = the minimum partition count

Then:

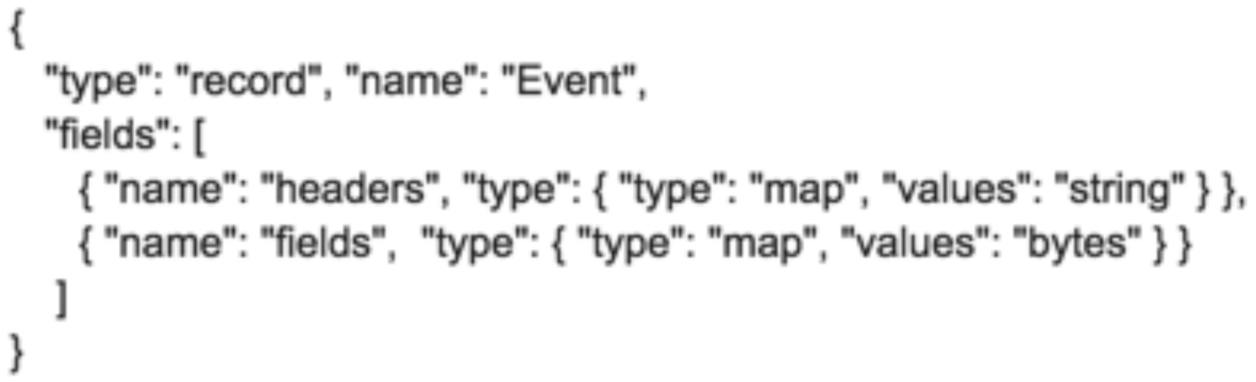
- pc >= max(tt/pt, tt/ct)

Choosing Partition Counts: Tools

- Kafka includes rudimentary benchmarking tools to help you get a rough estimate
  - kafka-producer-perft-test.sh (kafka.tools.ConsumerPerformance)
  - kafka-consumer-perf-test.sh (kafka.tools.ProducerPerformance)
  - kafka.tools.EndToEndLatency
    - Use with kafka-run-class.sh
- Nothing is more accurate than a real application
  - With real/representative data

How do I manage Schemas?

- A big topic with enough content for its own talk
- Options
  - Schema Registry
  - Source Controlled Dependency
  - Static "Envelop Schema”



```
{
   "type": "record", "name": "Event",
   "fields": [
      { "name": "headers", "type": { "type": "map", "values": "string" } },
      { "name": "fields",  "type": { "type": "map", "values": "bytes" } }
   ]
}
```

cloudera

Thank you