kafka

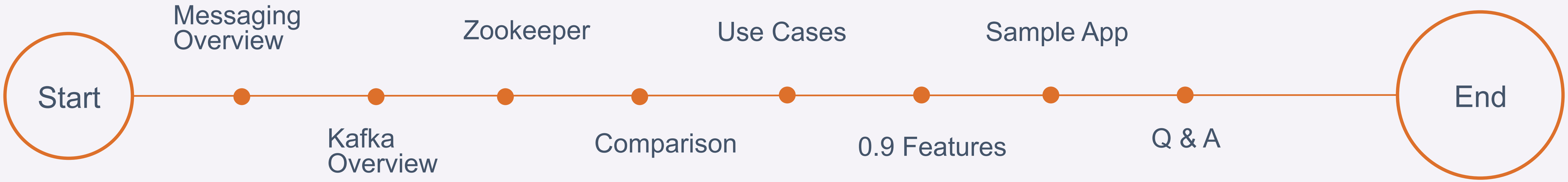
Introduction & Use Cases

Amita Mirajkar, Sunny Gupta
Clairvoyant India Pvt. Ltd.

CLAIRVOYANT

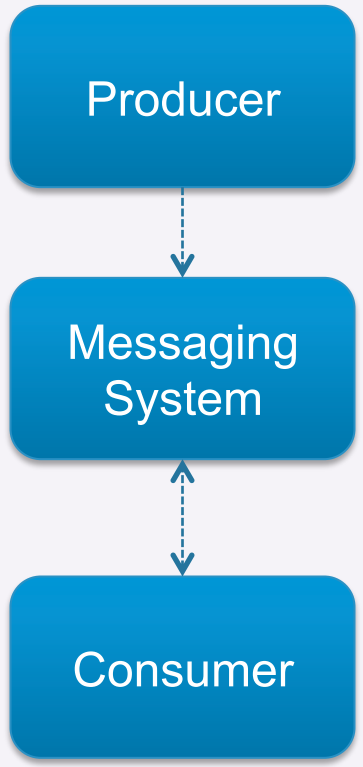
design engineer deliver

Presentation Agenda



Messaging Systems

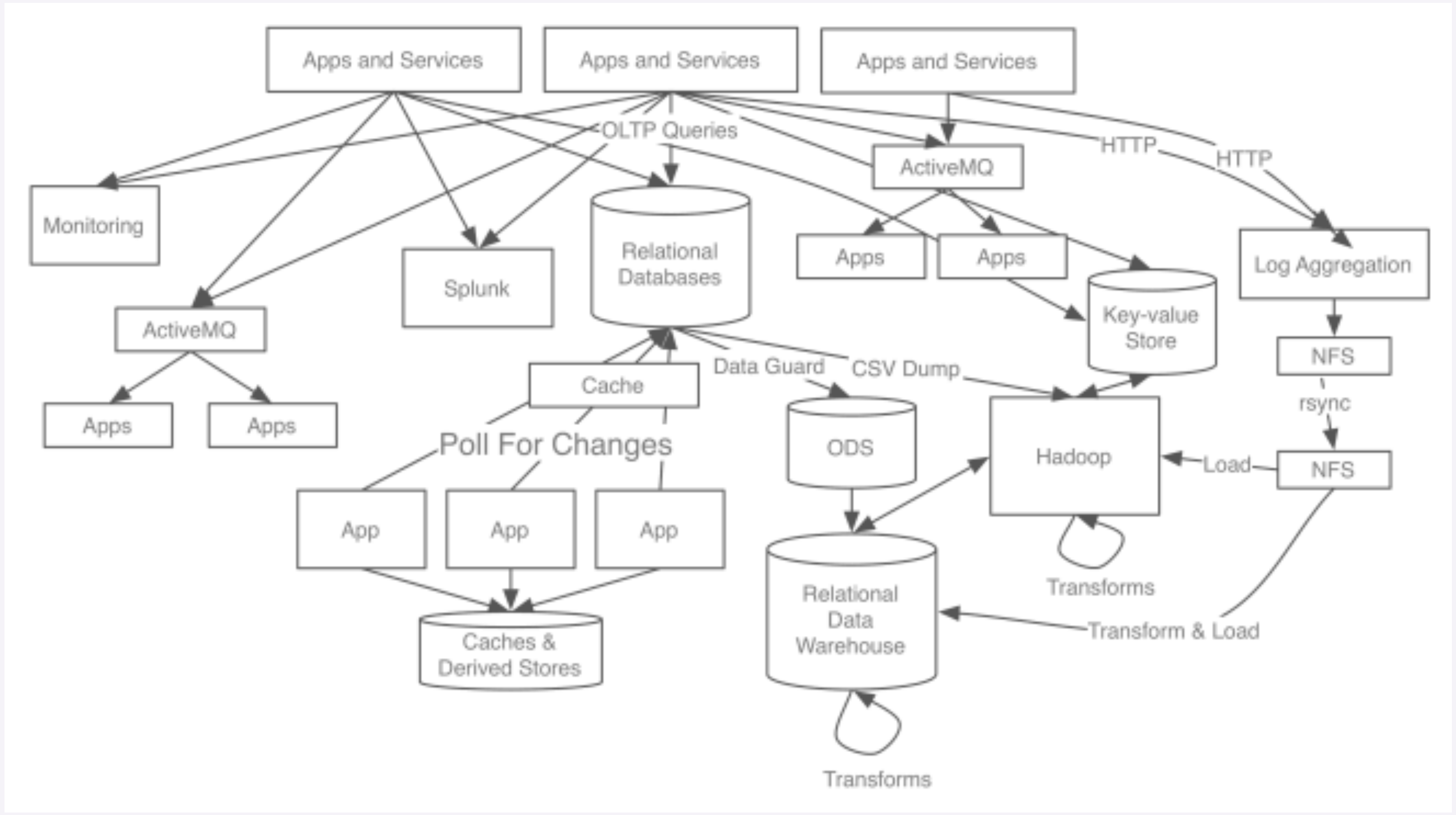
- **Asynchronous** communication between systems
- Some Use Cases
    - Web application – fast response to client and handle heavy processing tasks asynchronously
    - Balance load between workers
    - **Decouple** processing from data producers
- Models
    - Queuing: a pool of consumers may read from a server and each message goes to one of them
    - Publish – Subscribe: the message is broadcast to all consumers



Kafka

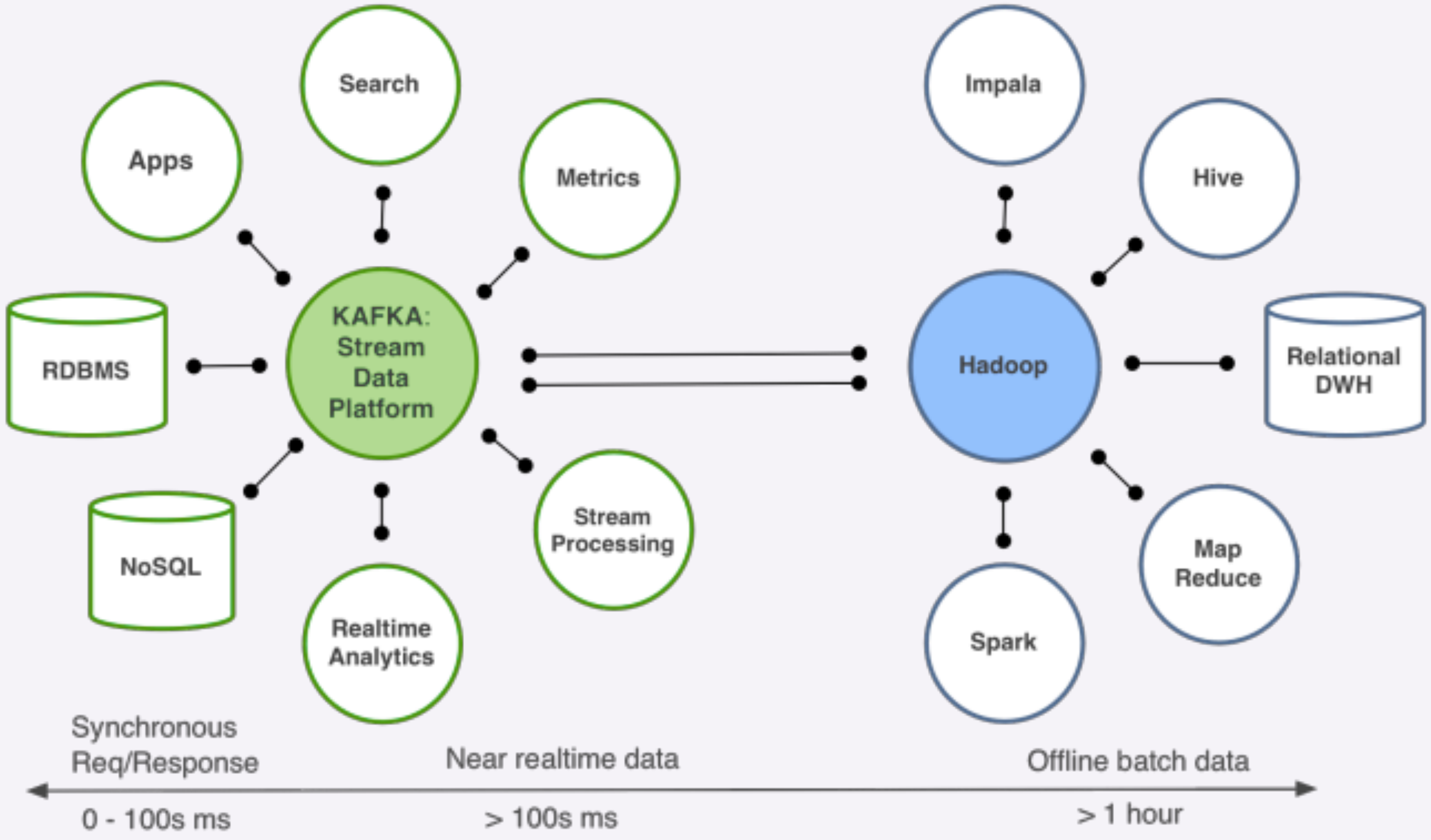
- Kafka is a open-source message broker project
- Distributed, replicated, scalable, durable, and gives high throughput
- Aim – “**central nervous system for data**”
- The design is heavily influenced by transaction logs
- Built at LinkedIn with a specific purpose in mind: to serve as a central repository of data streams

Motivation



At LinkedIn before Kafka – Complex setup with pipelines between different systems

Motivation



Creators of Kafka imagined something like this… **Stream Data Platform**

Kafka

- After Kafka in place, LinkedIn stats look great – as of March 2015 –
    - 800B messages produced / day – almost 175 TB of data
    - 1100 Kafka brokers organized in 60 clusters
  - As of Sep 2015… around 1.1 trillion a day...

- Written in Scala, open-sourced in 2011 under the Apache Software Foundation

- Apache top level project since 2012

Kafka Terminology

Kafka broker

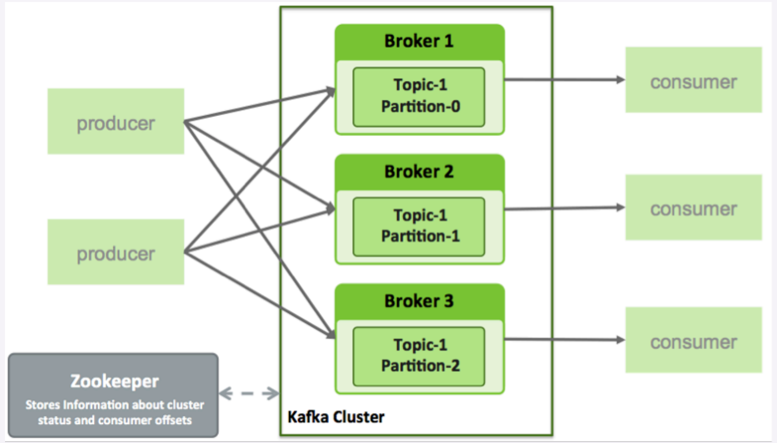
- Designed for HA - there are no master nodes. All nodes are interchangeable.
- Data is replicated.
- Messages are stored for configurable period of time

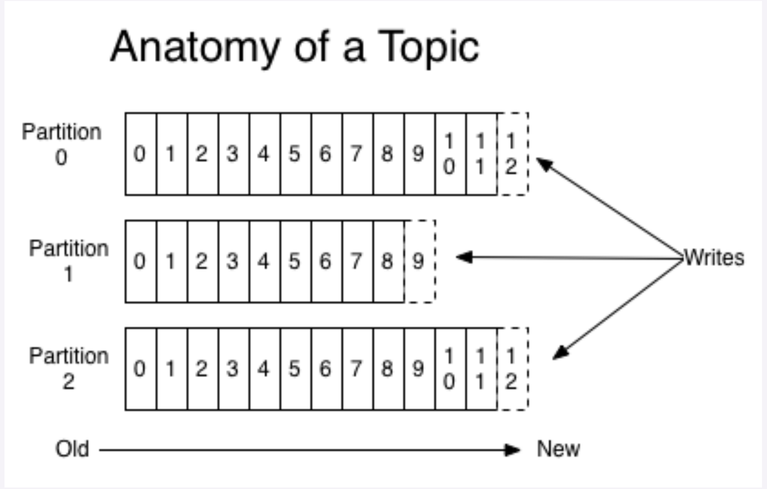
Topic

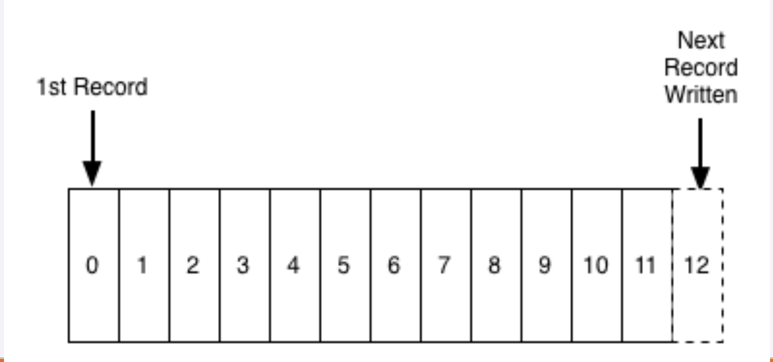
- A topic is a category or feed name to which messages are published.
- Topics are partitioned

Log

- Append Only
- Totally ordered sequence of records – ordered by time
- They record what happened and when

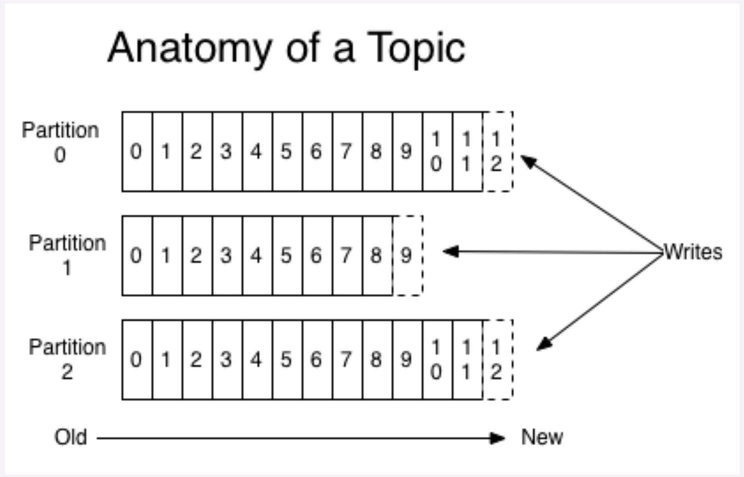






Kafka Terminology (cont.)

- Partitions
  - Each partition is an **ordered**, immutable sequence of messages that is continually appended to —a commit log
  - Each message in the partition is assigned a unique sequenced ID, its **offset**
  - More partitions allow greater **parallelism** for consumption
  - They allow the log to scale beyond a size that will fit on a single server. Each individual partition must fit on the servers that host it, but a topic can handle an arbitrary amount of data.
  - Number of partitions decide number of workers
  - Each partition has one server which acts as the "leader" and zero or more servers which act as "followers".
    - Leader handles all read and write requests for the partition.



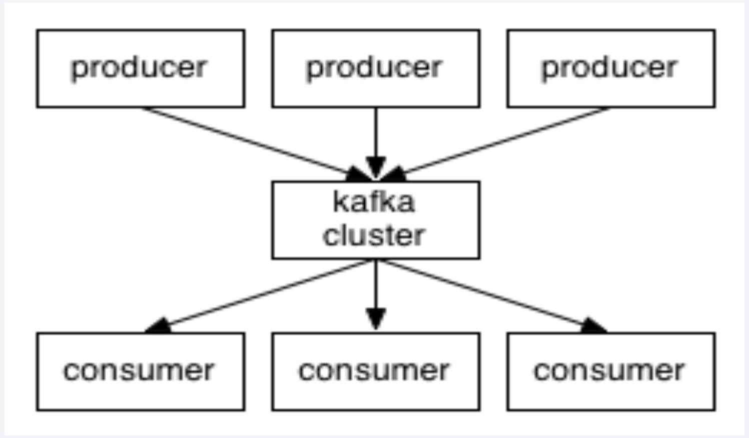
Kafka Terminology (cont.)

Producers

- Send messages to topics synchronously or asynchronously
- They **decide**
  - Partition / Key / none of these / Partitioner class
  - what sort of replication guarantees they want (acks setting)
  - batching and compressing

Consumers and Consumer Groups

- Consumer labels themselves with a consumer group name; and subscribe to one or more topics
- Consumers ***pull*** messages
- They control the **offset** read by them .. Can re-read without overhead on broker
- Each consumer in a consumer group will read messages from a unique subset of partitions in each topic they subscribe to, so each message is delivered to one consumer in the group, and all messages with the same key arrive at the same consumer



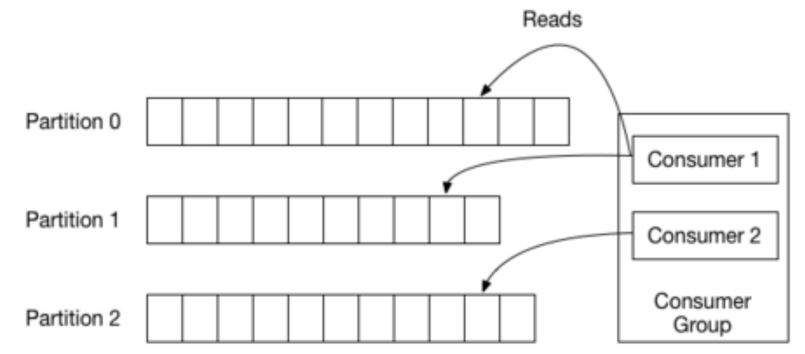
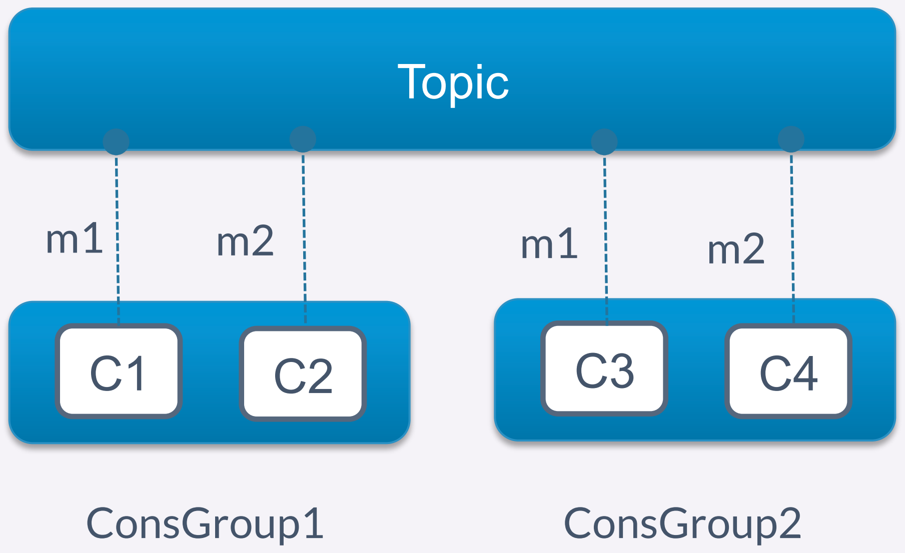


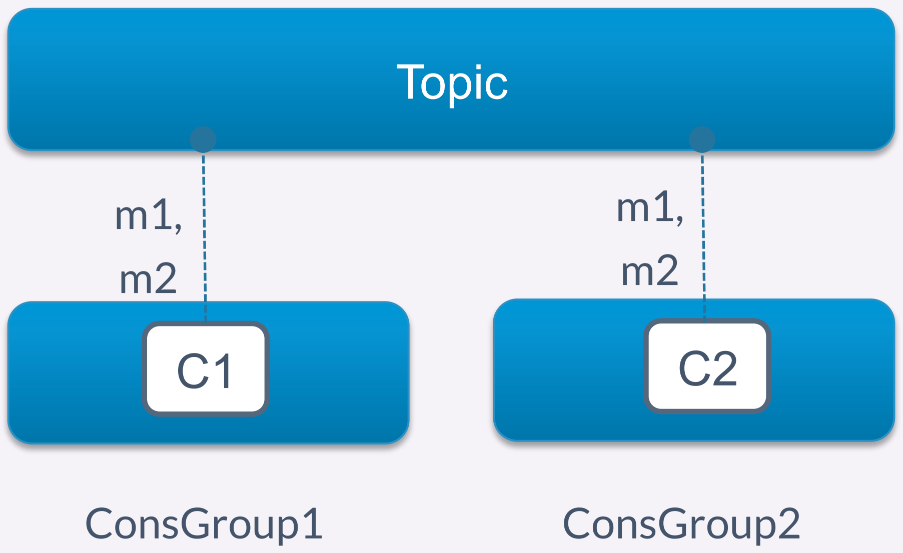
Figure 1: Consumer Group

Kafka Terminology – Consumer Groups

Queue model



Publish-subscribe model



Zookeeper

- ZooKeeper is a fast, highly available, fault tolerant, distributed **coordination service**
    - help distributed synchronization and
    - maintain configuration information
- Replicated: Like the distributed processes it coordinates, ZooKeeper itself is intended to be replicated over a sets of hosts called an ensemble.
- Role in kafka architecture
    - Coordinate cluster information
    - Store cluster metadata
    - Store consumer offsets

Differences with RabbitMQ

Feature	Kafka	JMS Message Broker; RabbitMQ
Dequeuing	cluster retains all published messages—whether or not they have been consumed—for a configurable period of time.	When consumer acknowledges
Consumer metadata	the only metadata retained on a per-consumer basis is offset.	consumer acknowledgments per message
Ordering	Strong ordering within a partition	Ordering of the messages is lost in the presence of parallel consumption. For workaround of "exclusive consumer" have to sacrifice parallelism
Batching / Streaming	Available for both producer and consumer – supports online and offline consumers	Consumers are mostly online
Scalability	Client centric	Broker centric
Complex routing	Needs to be programmed	Lot of options available with less work
Monitoring UI	Needs work	Decent web UI available

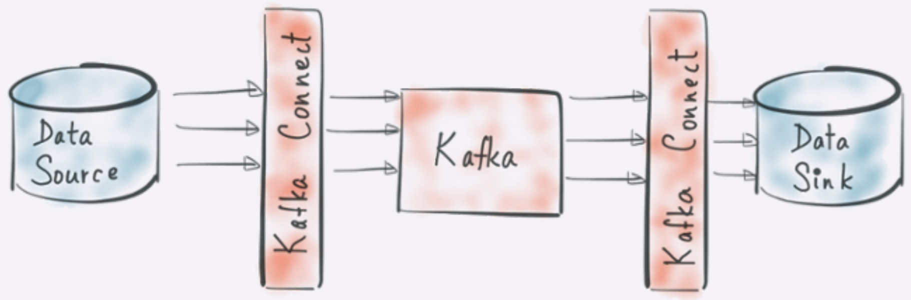
Common Use Cases

- Messaging
- Website Activity Tracking
  - The original use case for Kafka - Often very high volume –
  - (page views, searches, etc.) -> published to central topics -> subscribed by different consumers for various use cases - real-time processing, monitoring, and loading into Hadoop or offline processing and reporting.
- Log Aggregation
- Stream Processing
  - Collect data from various sources
  - Aggregate the data as soon as it arrives
  - Feed it to systems such as Hadoop/ DB/ other clients

Kafka 0.9 Features

- Security
  - authenticate users using either Kerberos or TLS client certificates
  - Unix-like permission system to control which user can access which data
  - encryption
- Kafka Connect
- User defined Quota
- New Consumer
  - New Java client
  - Group management facility
  - Faster rebalancing
  - Fully decouple clients from Zookeeper

KAFKA CONNECT



Bootstrapping

Bootstrapping for producers

1. Cycle through a list of "bootstrap" kafka urls until we find one we can connect to. Fetch cluster metadata.
2. Process fetch or produce requests, directing them to the appropriate broker based on the topic/partitions they send to or fetch from
3. If we get an appropriate error, refresh the metadata and try again.

Bootstrapping of consumers

1. On startup or on co-ordinator failover, the consumer sends a ConsumerMetadataRequest to any of the brokers in the bootstrap.brokers list -> receives the location of the co-ordinator for it's group.
2. The consumer connects to the co-ordinator and sends a HeartbeatRequest.
3. If no error is returned in the HeartbeatResponse, the consumer continues fetching data, for the list of partitions it last owned, without interruption.

SAMPLE APPLICATION

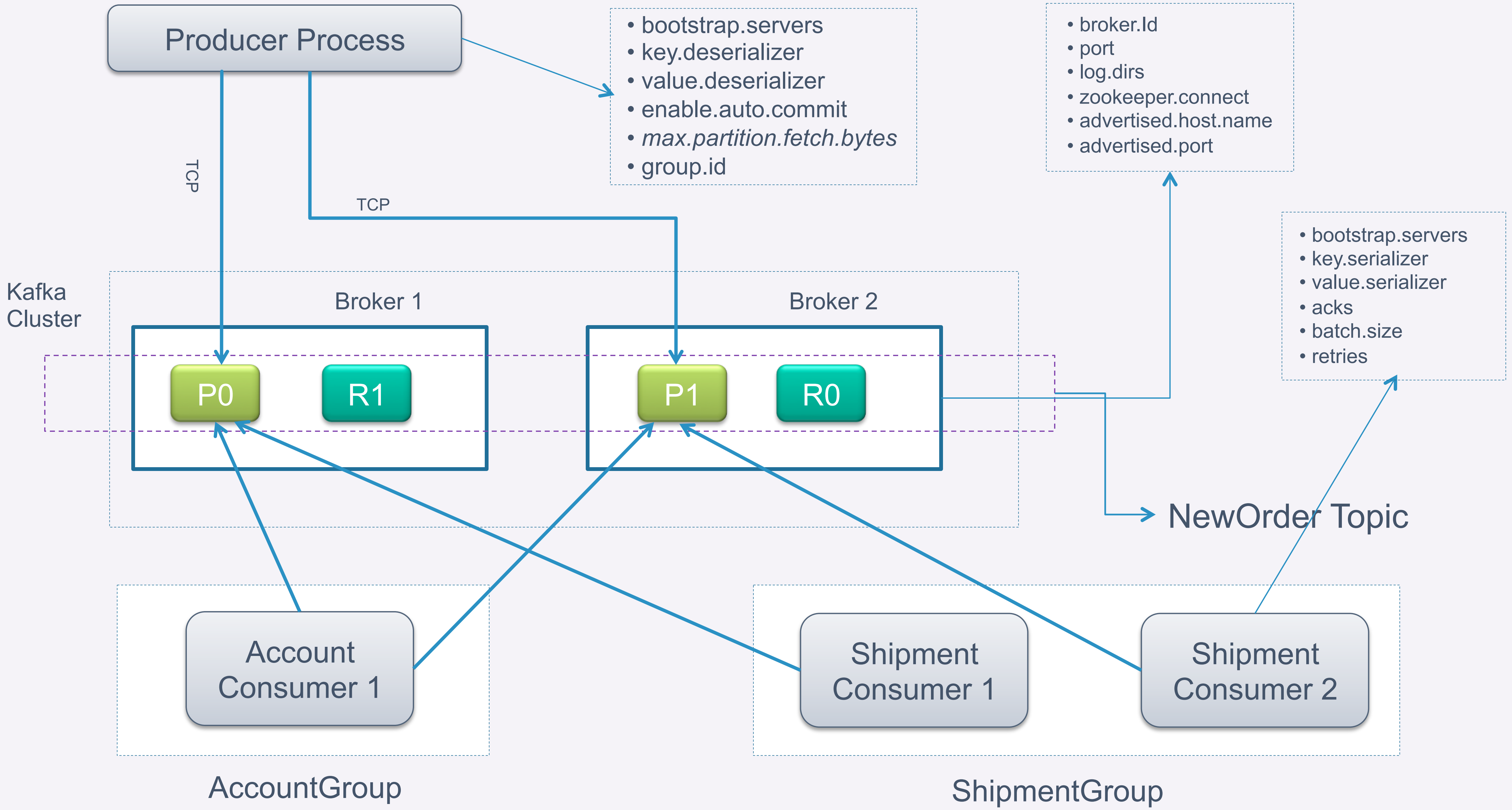
Sample Application

- E shopping system – simplified scenario
  - Supports shipping in two cities
  - Once order is placed we need to handle payment and shipping
  - Shipping system allows efficiency if requests are grouped by city
  - See simple architecture diagram in next slide and check out the code

In demo application, we will cover:

- Zookeeper config
- Broker config
  - Start two brokers
  - Create Topic and describe / list
- Producer config
- Message delivery semantics
- Consumer config
- Consumer Rebalancing

- Sample application code: https://github.com/teamclairvoyant/meetup-docs/tree/master/Meetup-Kafka



- bootstrap.servers
- key.deserializer
- value.deserializer
- enable.auto.commit
- *max.partition.fetch.bytes*
- group.id

- broker.Id
- port
- log.dirs
- zookeeper.connect
- advertised.host.name
- advertised.port

- bootstrap.servers
- key.serializer
- value.serializer
- acks
- batch.size
- retries

CLAIRVOYANT

QUESTIONS?

THANK YOU

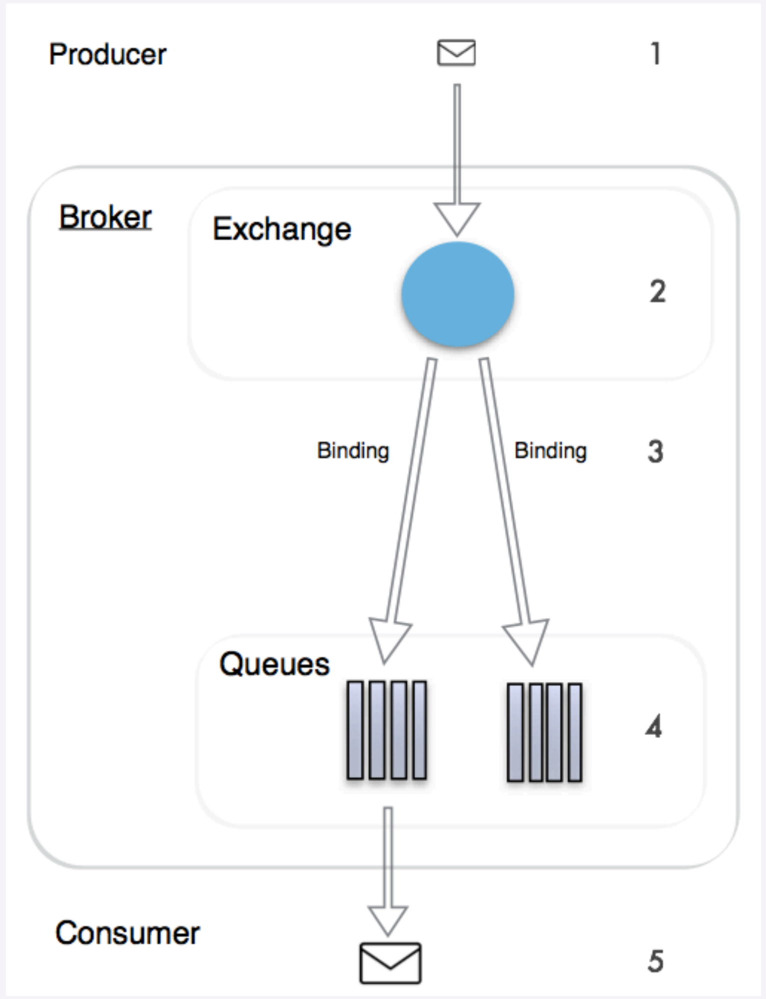
BACKUP SLIDES

References / Good Reads

- http://www.confluent.io/blog/stream-data-platform-1/
- http://kafka.apache.org/documentation.html
- http://www.infoq.com/articles/apache-kafka
- http://blog.cloudera.com/blog/2014/09/apache-kafka-for-beginners/
- http://www.confluent.io/blog/tutorial-getting-started-with-the-new-apache-kafka-0.9-consumer-client
- https://cwiki.apache.org/confluence/display/KAFKA/A+Guide+To+The+Kafka+Protocol
- https://cwiki.apache.org/confluence/display/KAFKA/System+Tools
- https://zookeeper.apache.org/doc/r3.3.2/zookeeperOver.html#ch_DesignOverview
- http://blog.cloudera.com/blog/2014/11/flafka-apache-flume-meets-apache-kafka-for-event-processing/
- http://www.slideshare.net/wangxia5/netflix-kafka

RabbitMQ

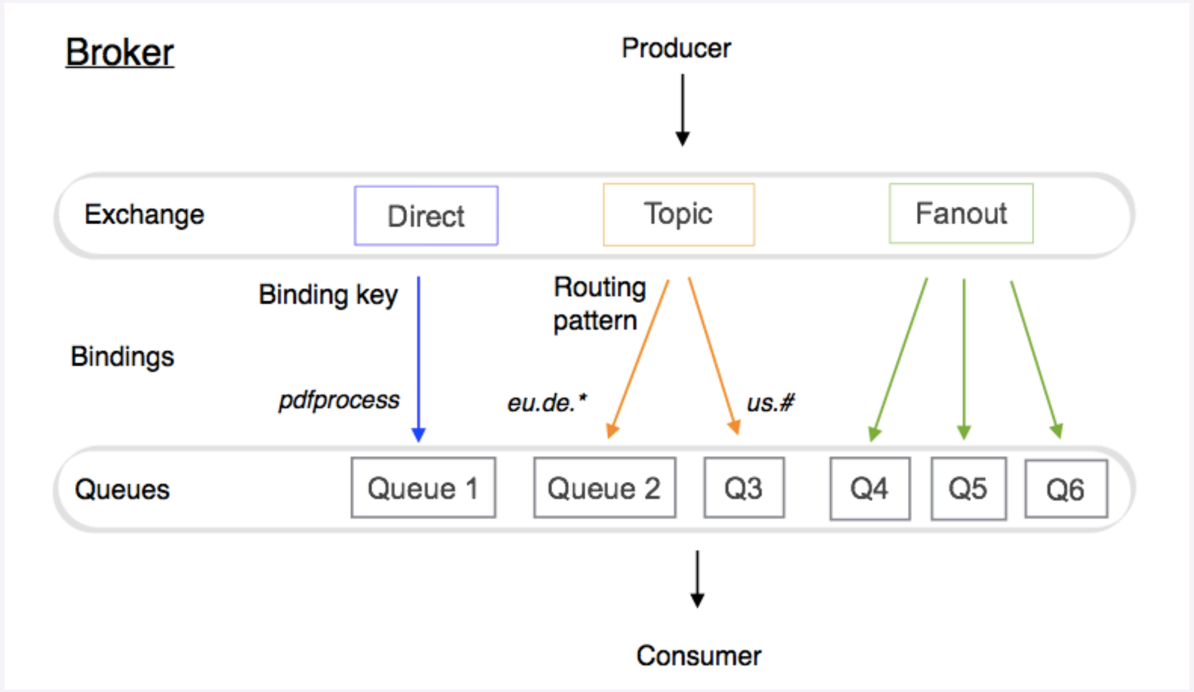
- Proven Message Broker uses *Advanced Message Queuing Protocol* (AMQP) for messaging.
- Message flow & concepts in RabbitMQ
  - The **producer** publishes a **message**
  - The **exchange** receives and routes the message in to the queues
  - **Routing** can be based on different message attributes such as routing key, depending on the exchange type
  - **Binding** is a link between an exchange and a queue
  - The messages stays in the **queue** until they are handled by a consumer
  - The **consumer** handles the message.
  - **Channel**: a virtual connection inside a connection. When you are publishing or consuming messages or subscribing to a queue is it all done over a channel



RabbitMQ (cont.)

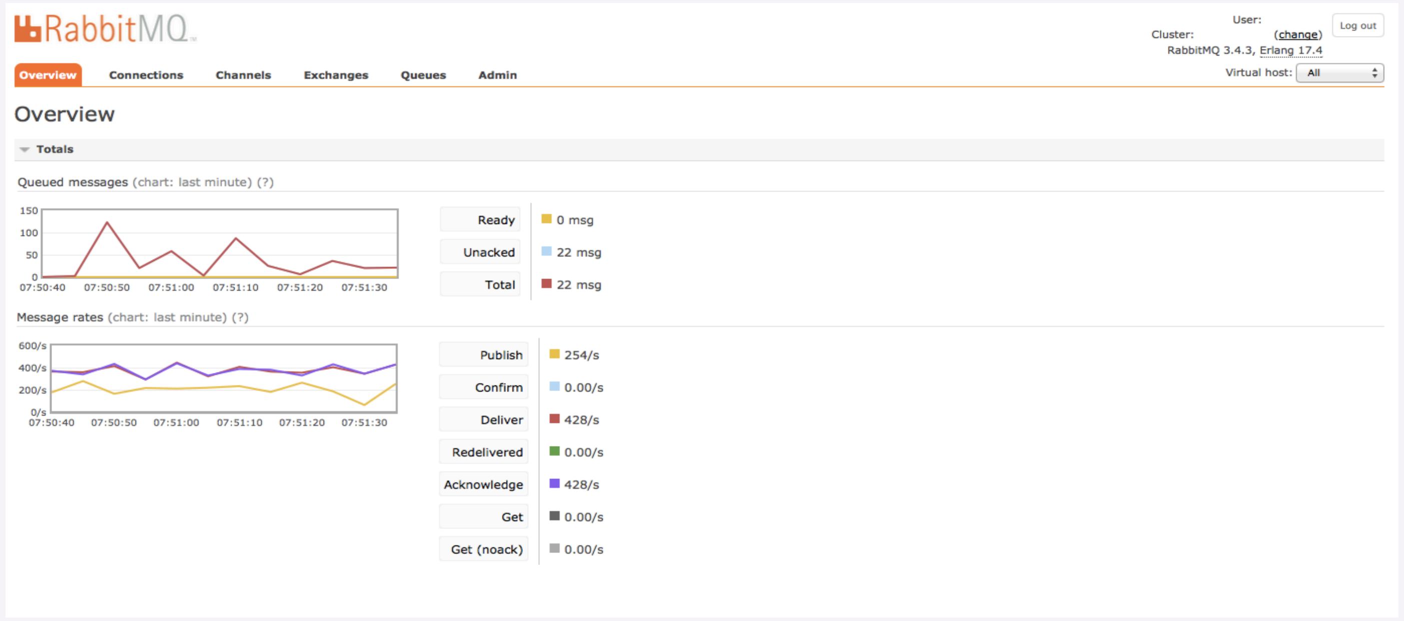
- Types of Exchange
  - **Direct**: delivers messages to queues based on a message routing key:

    Queues' binding key == routing key of the message
  - **Fanout**: routes messages to all of the queues that are bound to it.
  - **Topic**: does a wildcard match between the routing key and the routing pattern specified in the binding.
  - **Headers:** uses the message header attributes for routing.
- CloudAMQP
  - hosted RabbitMQ solution, just sign up for an account and create an instance. You do not need to set up and install RabbitMQ or care about cluster handling



RabbitMQ (cont.)

- Management and Monitoring
  - Nice web UI for management and monitoring of your RabbitMQ server.
  - Allows to handle, create, delete and list queues, monitor queue length, check message rate, change and add users permissions, etc.



Overview

Totals

Queued messages (chart: last minute) (?)

Ready	🟨 0 msg
Unacked	🟦 22 msg
Total	🟥 22 msg

Message rates (chart: last minute) (?)

Publish	🟨 254/s
Confirm	🟦 0.00/s
Deliver	🟥 428/s
Redelivered	🟩 0.00/s
Acknowledge	🟪 428/s
Get	⬛ 0.00/s
Get (noack)	⬜ 0.00/s

Upgrading from 0.8.0, 0.8.1.X or 0.8.2.X to 0.9.0.0

- 0.9.0.0 has potential breaking changes (please review before upgrading) and an inter-broker protocol change from previous versions.
- Java 1.6 and Scala 2.9 is no longer supported
- http://kafka.apache.org/documentation.html
- Kafka consumers in earlier releases store their offsets by default in ZooKeeper. It is possible to migrate these consumers to commit offsets into Kafka by following some steps

Kafka Terminology (cont.)

- Protocol
  - These requests to publish or fetch data must be sent to the broker that is currently acting as the **leader for a given partition.** This condition is enforced by the broker, so a request for a particular partition to the wrong broker will result in an the NotLeaderForPartition error code
  - All Kafka brokers can answer a **metadata request** that describes the current state of the cluster:
    - what topics there are
    - which partitions those topics have
    - which broker is the leader for those partitions
    - the host and port information for these brokers
  - Good explanation: https://cwiki.apache.org/confluence/display/KAFKA/A+Guide+To+The+Kafka+Protocol

Kafka Adoption

Apache Kafka has become a popular messaging system in a short period of time with a number of organizations like

- LinkedIn
- Tumblr
- PayPal
- Cisco
- Box
- Airbnb
- Netflix
- Square
- Spotify
- Pinterest
- Uber
- Goldman Sachs
- Yahoo and Twitter among others using it in production systems

THANK YOU