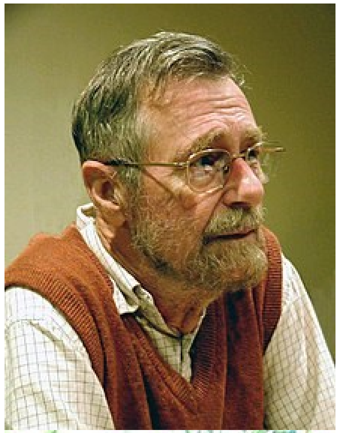
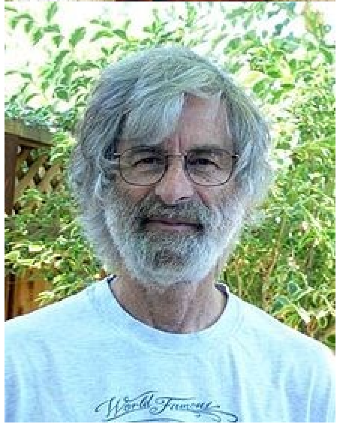
Distributed Systems

Dijkstra and Lamport





- Edsger W. Dijkstra: was a Dutch systems scientist, programmer, software engineer, science essayist,[9] and early pioneer in computing science.[10] He held the Schlumberger Centennial Chair in Computer Sciences at the University of Texas at Austin from 1984 until his retirement in 1999.
- **Leslie B. Lamport** is an American computer scientist. Lamport is best known for his seminal work in distributed systems and as the initial developer of the document preparation system LaTeX. Leslie Lamport was the winner of the 2013 Turing Lamport worked as SRI International from 1977 to 1985, and Digital Equipment Corporation and joined Microsoft Research

What is a Distributed System?

- A Collection of independent computer that appear to the users of the system as a single coherent computer.

Characteristics of DS

- The computers operate concurrently
- The computers fail independently
- The computers do not share a global clock

Examples

- What is the biggest example of DS
- WWW

Advantages

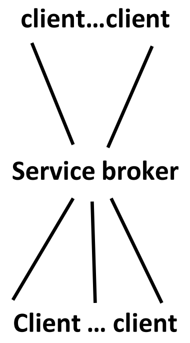
- A collection of microprocessors offer a better price/performance
- If one machine fails, the whole system can still survive
- Performance improves through load distribution
- Incremental growth

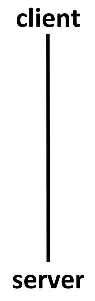
Technology Stack

- DB: Relational / Mongo DB, Cassandra, Riak, HDFS, etc.
- Computation: Hadoop, Spark, Storm
- Synchronization: NTP, Vector clocks
- Consensus: Paxos, Zookeeper
- Messaging: Kafka, RabbitMQ, etc.

Messaging

- Loosely Coupling systems
- Messages are consumed by subscribers and created by producers
- Messages are persisted for a short period of time
- Organized into topics
- Processed by brokers
- Preserve message ordering





Separation of Concerns

- Separation of concerns is the software engineering principle that each component should have a single small job to do so it can do it well
- In distributed systems, there are at least three concerns having to do with remote services: what to request, where to do it, how to ask for it.
- What to request: application programmer must figure this out, e.g. access customer database.
- Where to do it: application programmer should not need to know where, because this adds complexity + if location changes, application break.
- How to ask for it: want a uniform interface.

Message Queue

Part of contents are from John O’Hara

Message Queue

- Message: an immutable array of bytes
- Topic: a feed of messages
- Producer: a process that publishes messages to a topic
- Consumer: a single-threaded process that subscribes to a topic
- Broker: one of servers that comprise a cluster

Message Passing

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
- **Non-blocking** is considered **asynchronous**

Synchronous Client/Server

- The most straightforward interaction between components is the request/response
  - closely resembles the way we program
  - the model is simple and intuitive
  - well supported by RPC and the systems built around RPC
- Synchronous interaction requires both parties to be “on-line”.
- The caller must wait until the response comes back. What if a receiver does not exist?

Solutions

- Synchronous Interaction
  - Service replication and load balancing
  - Enable complex interactions with some execution guarantee
- Asynchronous Interaction
  - the caller sends a message that gets stored somewhere until the receiver reads it and sends a response.
- Asynchronous interaction can take place in two forms:
  - non-blocking invocation (similar to batch jobs)
  - persistent queues

Failure Semantics

- At-most-once delivery
- At-least-once delivery
- Exactly-once delivery

Failure Semantics

- A great deal of the functionality built around RPC tries to address the problem of failure semantics,
- Exactly-once semantics solves this problem but it has hidden costs:
- it implies atomicity in all operations
- the server must support some form of 2PC
- it usually requires a coordinator to oversee the interaction (the coordinator may also fail)

Publish/Subscribe

- Standard client/server architectures and queuing systems assume the client and the server know each other
- a service publishes messages/events of given type
- clients subscribe to different types of messages/events
- when a service publishes an event, the system looks at a table of subscriptions and forwards the event to the interested clients;

Apache Kafka

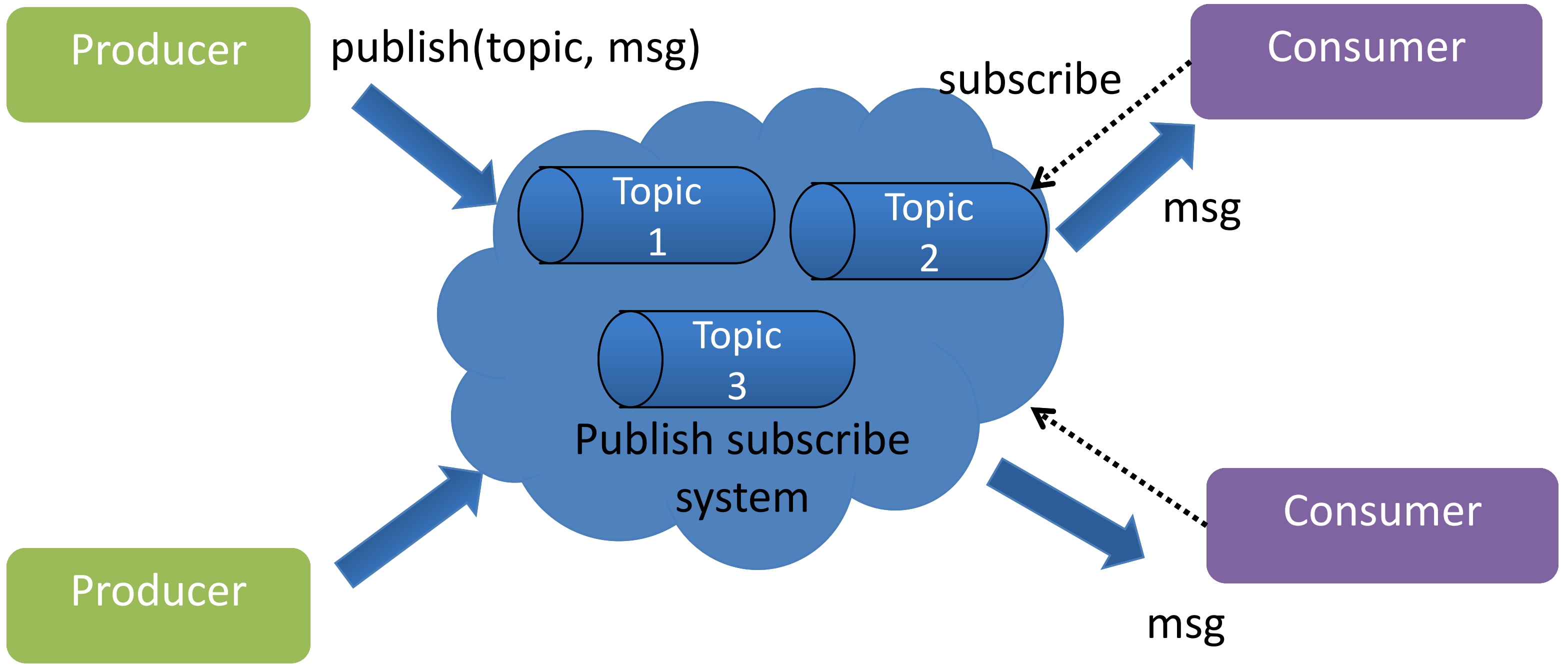
A high-throughput distributed messaging system

What is Apache Kafka?

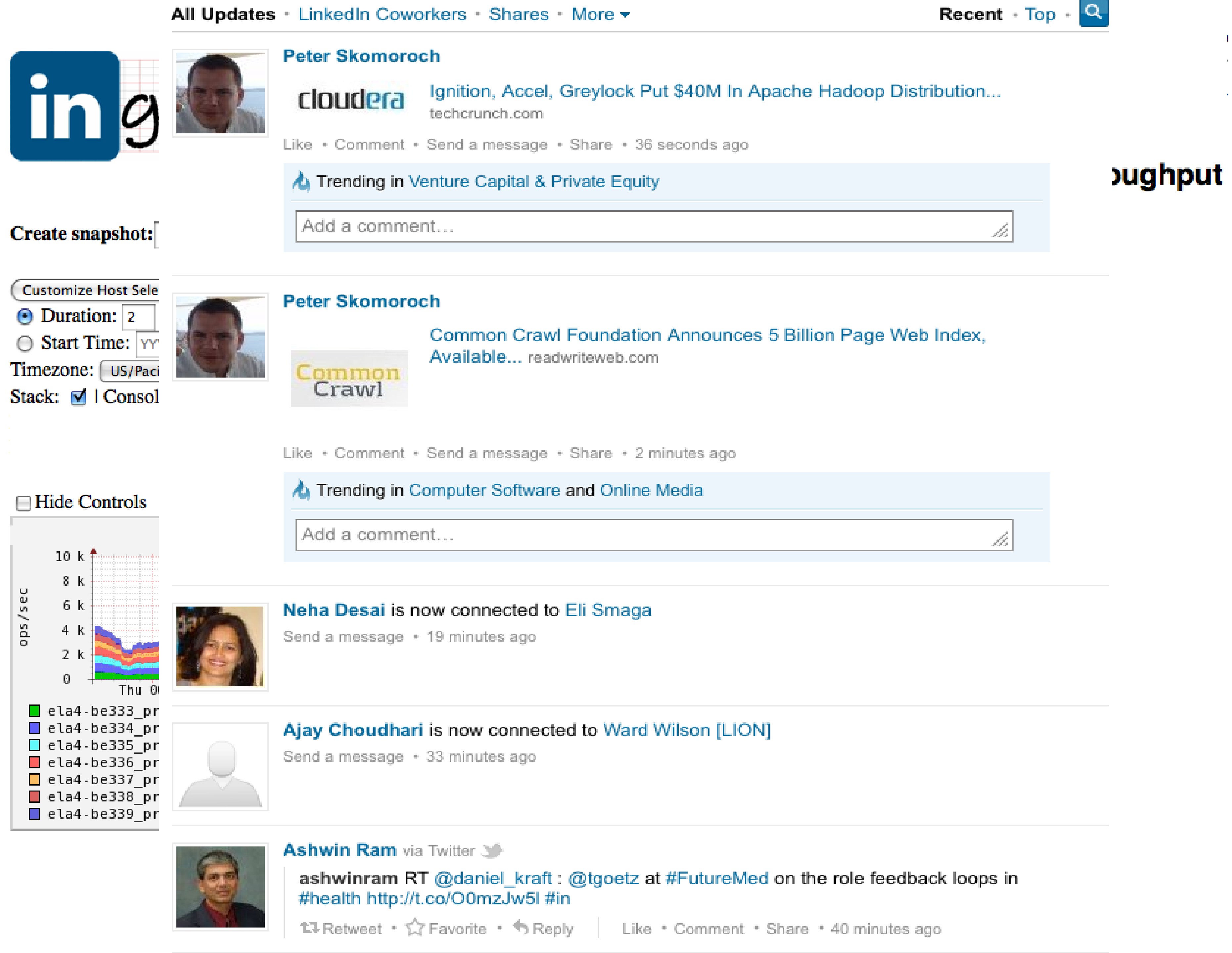
- **Distributed**, high-throughput, **pub-sub** messaging system
  - **Fast, Scalable, Durable**
- Main use cases:
  - log aggregation, real-time processing, monitoring, queueing
- Originally developed by LinkedIn
- Implemented in Scala/Java
- Top level Apache project since 2012: http://kafka.apache.org/

Source: Johan Lundahl

What is pub sub ?



Who is using Kafka



All Updates · LinkedIn Coworkers · Shares · More

Recent · Top ·

Peter Skomoroch

Ignition, Accel, Greylock Put $40M In Apache Hadoop Distribution...
techcrunch.com

Like · Comment · Send a message · Share · 36 seconds ago

Trending in Venture Capital & Private Equity

Add a comment...

Peter Skomoroch

Common Crawl Foundation Announces 5 Billion Page Web Index, Available... readwriteweb.com

Like · Comment · Send a message · Share · 2 minutes ago

Trending in Computer Software and Online Media

Add a comment...

Neha Desai is now connected to Eli Smaga

Send a message · 19 minutes ago

Ajay Choudhari is now connected to Ward Wilson [LION]

Send a message · 33 minutes ago

Ashwin Ram via Twitter

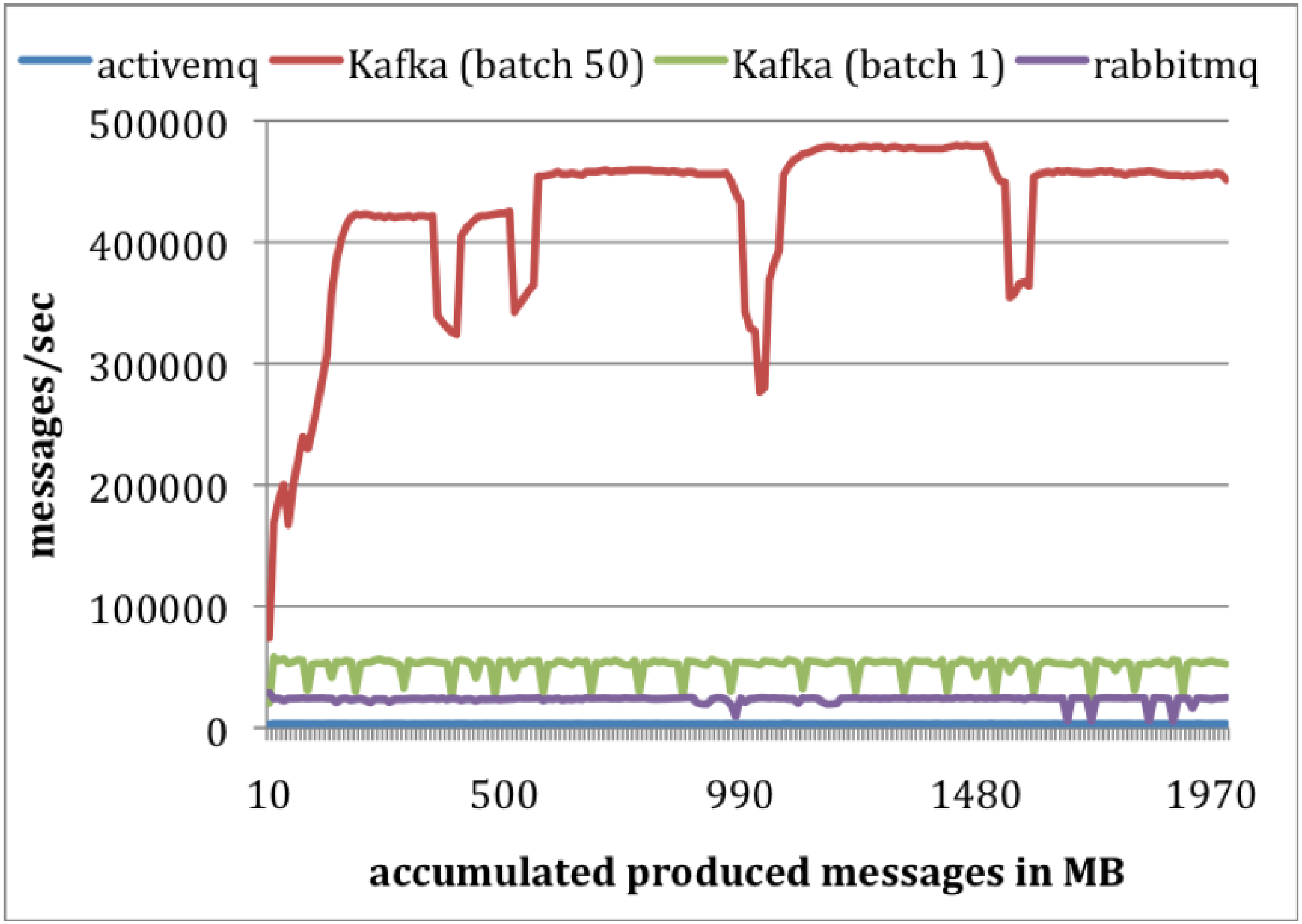
ashwinram RT @daniel_kraft : @tgoetz at #FutureMed on the role feedback loops in #health http://t.co/O0mzJw5l #in

Retweet · Favorite · Reply | Like · Comment · Share · 40 minutes ago

🟩	ela4-be333_pr
🟦	ela4-be334_pr
🟦	ela4-be335_pr
🟥	ela4-be336_pr
🟧	ela4-be337_pr
🟥	ela4-be338_pr
🟦	ela4-be339_pr

Volume

- 20B events/day
- 3 terabytes/day
- 150K events/sec


activemq
Kafka (batch 50)
Kafka (batch 1)
rabbitmq
messages/sec
500000
400000
300000
200000
100000
0
10
500
990
1480
1970
accumulated produced messages in MB

Kafka

- Kafka was originally developed at LinkedIn in 2011
- Nowadays it is a whole platform, allowing you to redundantly store vast amounts of data, have a message bus with huge throughput(millions/sec) and use real-time stream processing on the data that goes through it all at once.
- A distributed system is one which is split into multiple running machines, all of which work together in a cluster to appear as one single node to the end user.
- Kafka is distributed in the sense that it stores, receives and sends messages on different nodes (called brokers).
- Kafka is a distributed, horizontally-scalable, fault-tolerant, commit log.

Horizontally-scalable

- Scale your application by adding new identical machines.
- Adding a new machine does not require downtime nor are there any limits to the amount of machines you can have in your cluster. The catch is that not all systems support horizontal scalability, as they are not designed to work in a cluster and those that are are usually more complex to work with

Replication

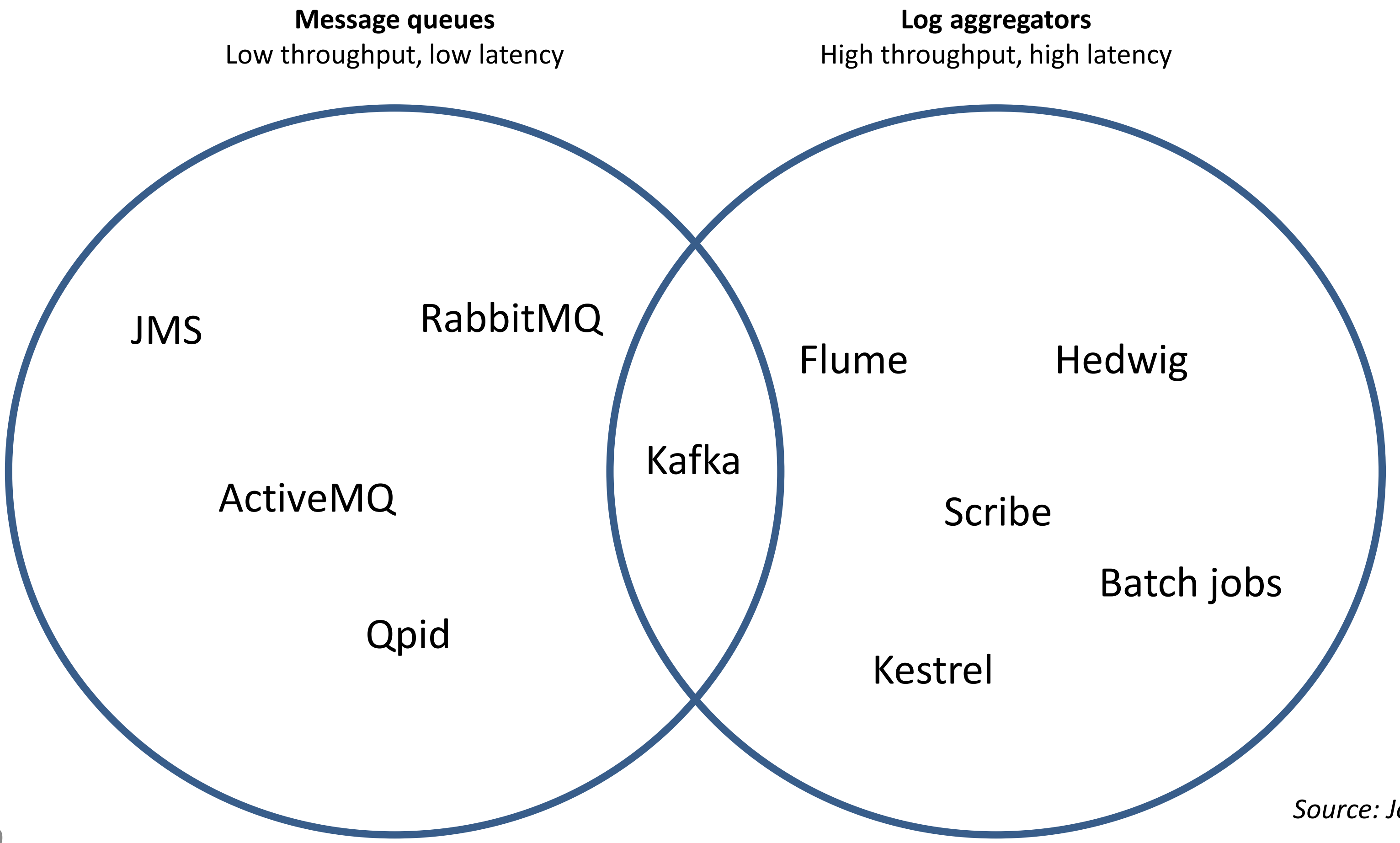
- Brokers can fail
- Configurable in-sync replica sets
- One leader, n-1 followers
- Only the leader communicates with clients
- When the leader fails, a new leader is elected
- Distributed systems are designed in such a way to accommodate failures in a configurable way. In a 5-node Kafka cluster, you can have it continue working even if 2 of the nodes are down.

Kafka

- Widely deployed
- JVM based (written in Scala)
- Native Java and Scala APIs
- Binary wire protocol
- Start at three nodes and grow

Comparison to other messaging systems

- Traditional: JMS, xxxMQ/AMQP
- New gen: Kestrel, Scribe, Flume, Kafka

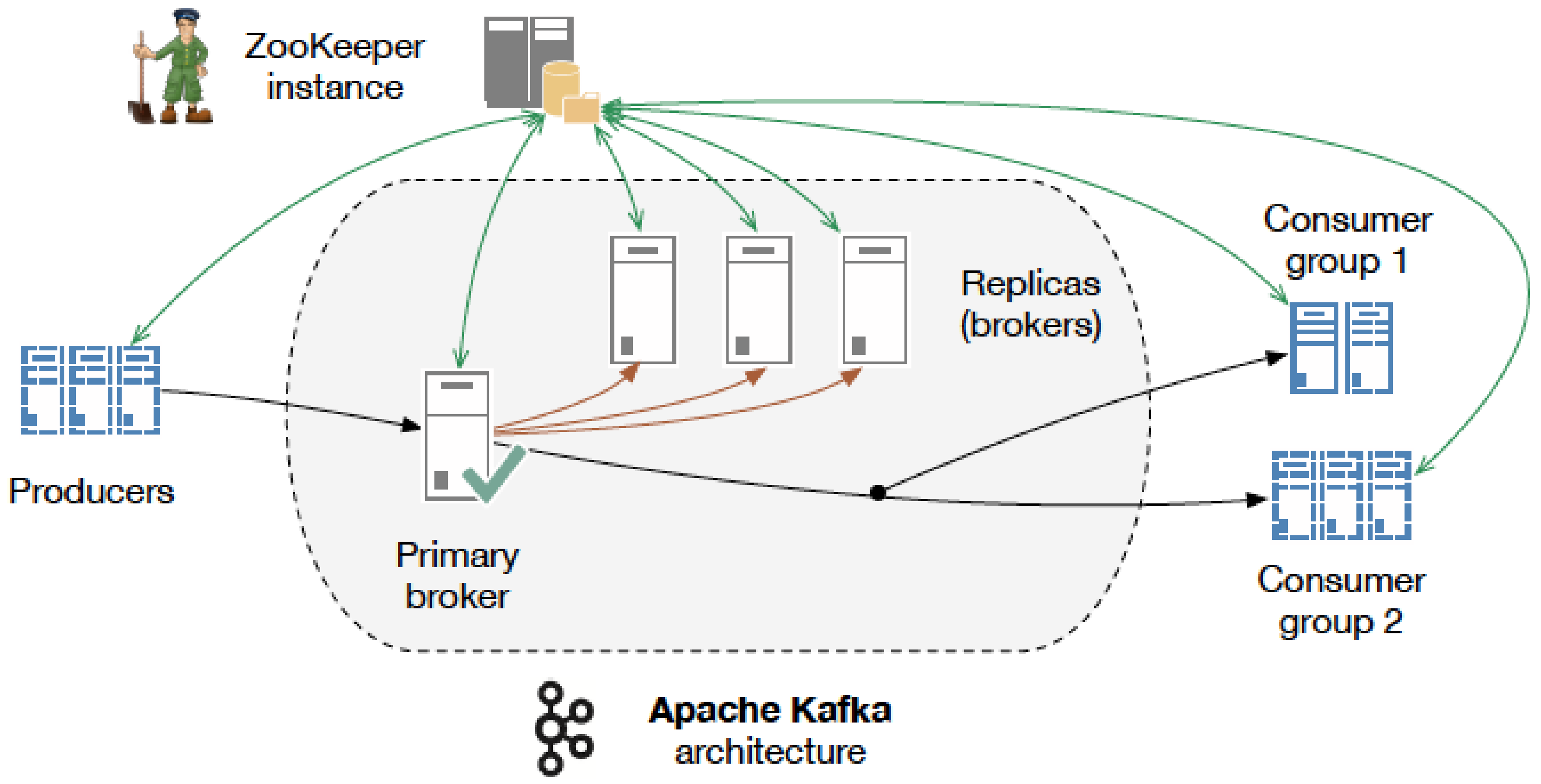


Source: Johan Lundahl

Kafka Architecture

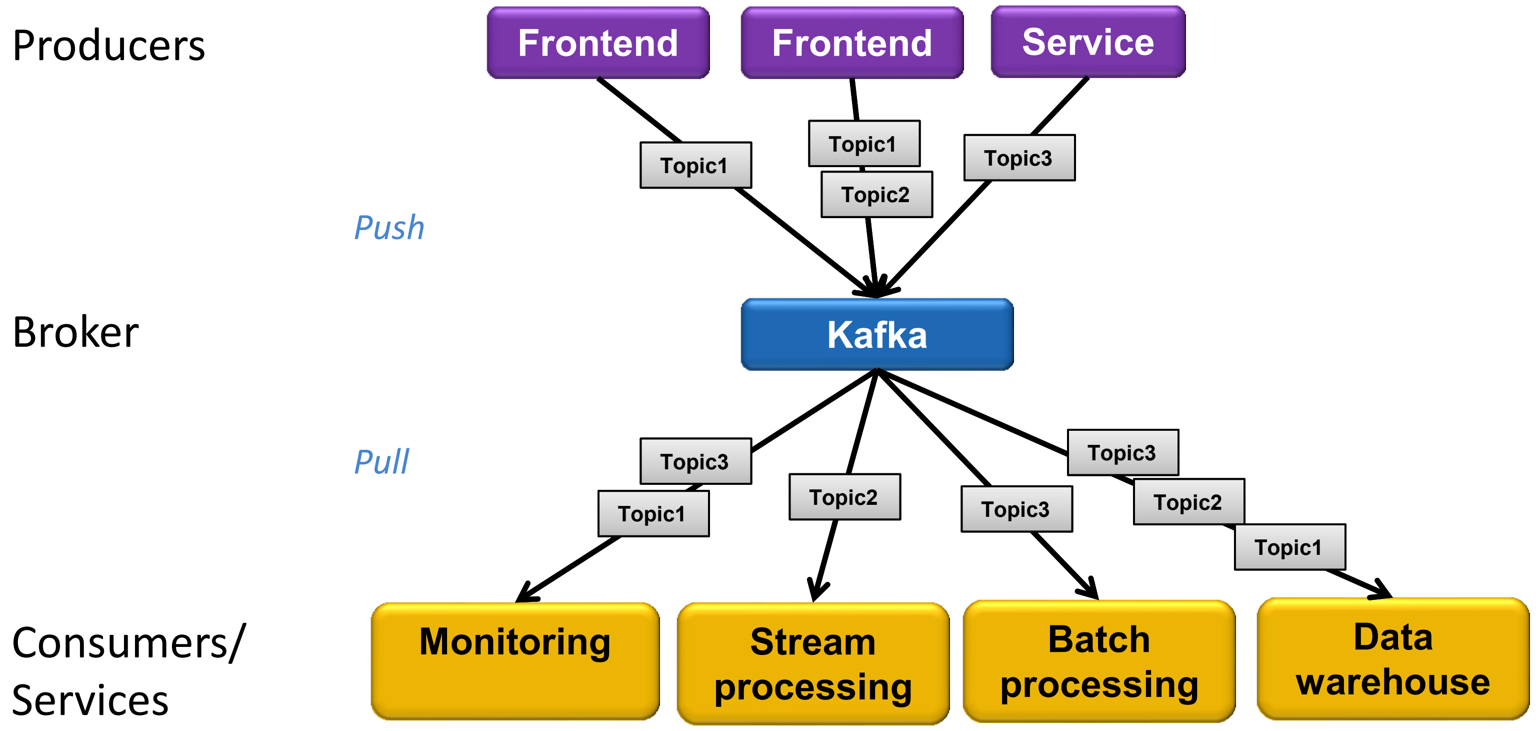
- Kafka consists of Records, Topics, Consumers, Producers, Brokers, Logs, Partitions, and Clusters.
- Records can have key (optional), value and timestamp.
- Kafka Records are immutable.
- A Kafka Topic is a stream of records ("/orders", "/user-signups"). You can think of a Topic as a feed name.
- A topic has a Log which is the topic’s storage on disk. A Topic Log is broken up into partitions and segments.
- The Kafka Producer API is used to produce streams of data records.
- The Kafka Consumer API is used to consume a stream of records from Kafka.
- A Broker is a Kafka server that runs in a Kafka Cluster. Kafka Brokers form a cluster. The Kafka Cluster consists of many Kafka Brokers on many servers. Broker sometimes refer to more of a logical system or as Kafka as a whole.

Kafka



Source: Kafka

Kafka concepts

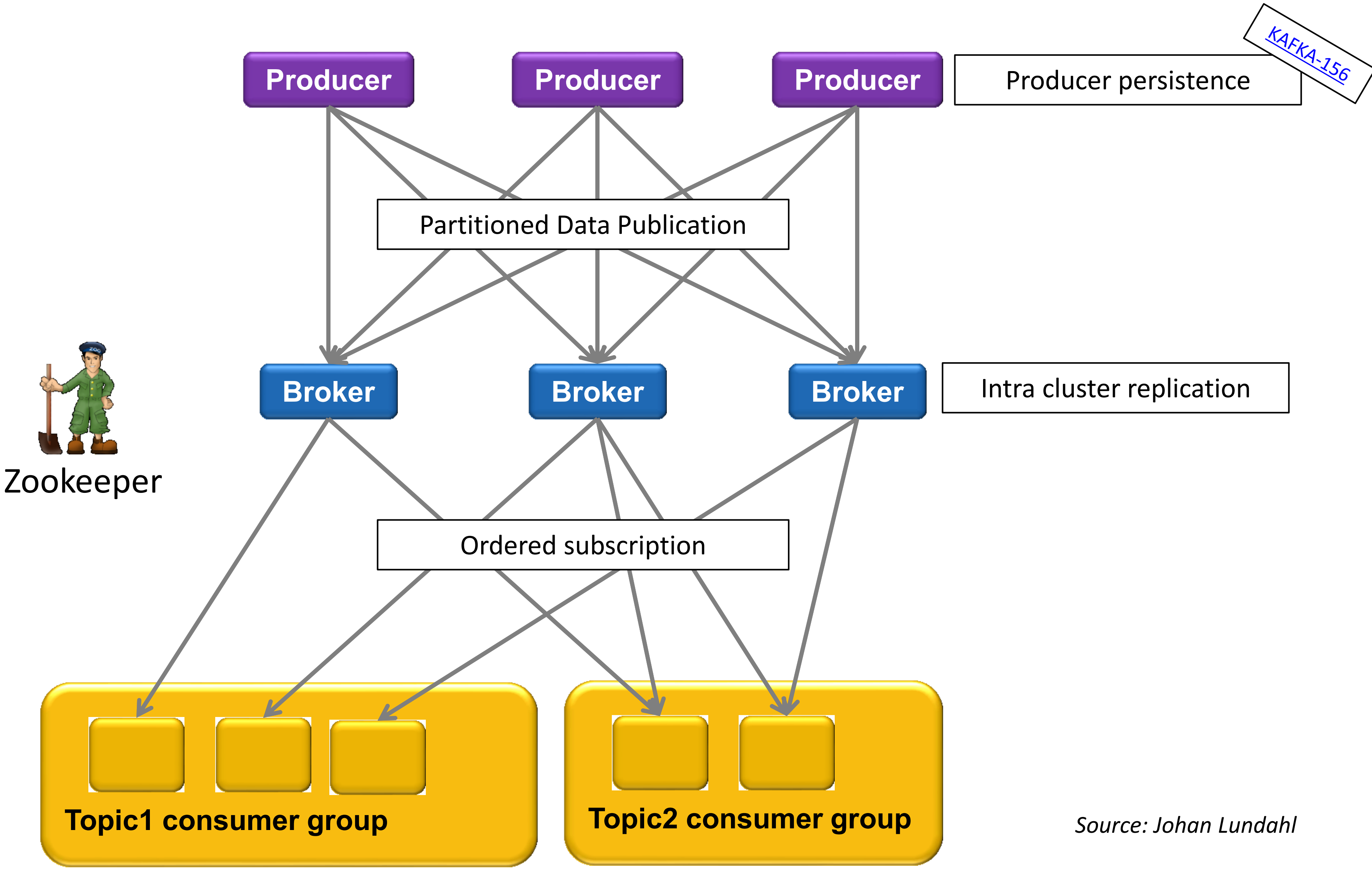


Source: Kafka

Zookeeper

- Required part of Kafka architecture
- Producers use it to find partitions and replication information
- Consumers use it to track current index
- Kafka uses Zookeeper to do leadership election of Kafka Broker and Topic Partition pairs. Kafka uses Zookeeper to manage service discovery for Kafka Brokers that form the cluster. Zookeeper sends changes of the topology to Kafka, so each node in the cluster knows when a new broker joined, a Broker died, a topic was removed or a topic was added, etc. Zookeeper provides an in-sync view of Kafka Cluster configuration.

Distributed model



Source: Johan Lundahl

Zookeeper

- Zookeeper is a centralized service for mainlining configuration information naming, providing distributed synchornization and providing group services

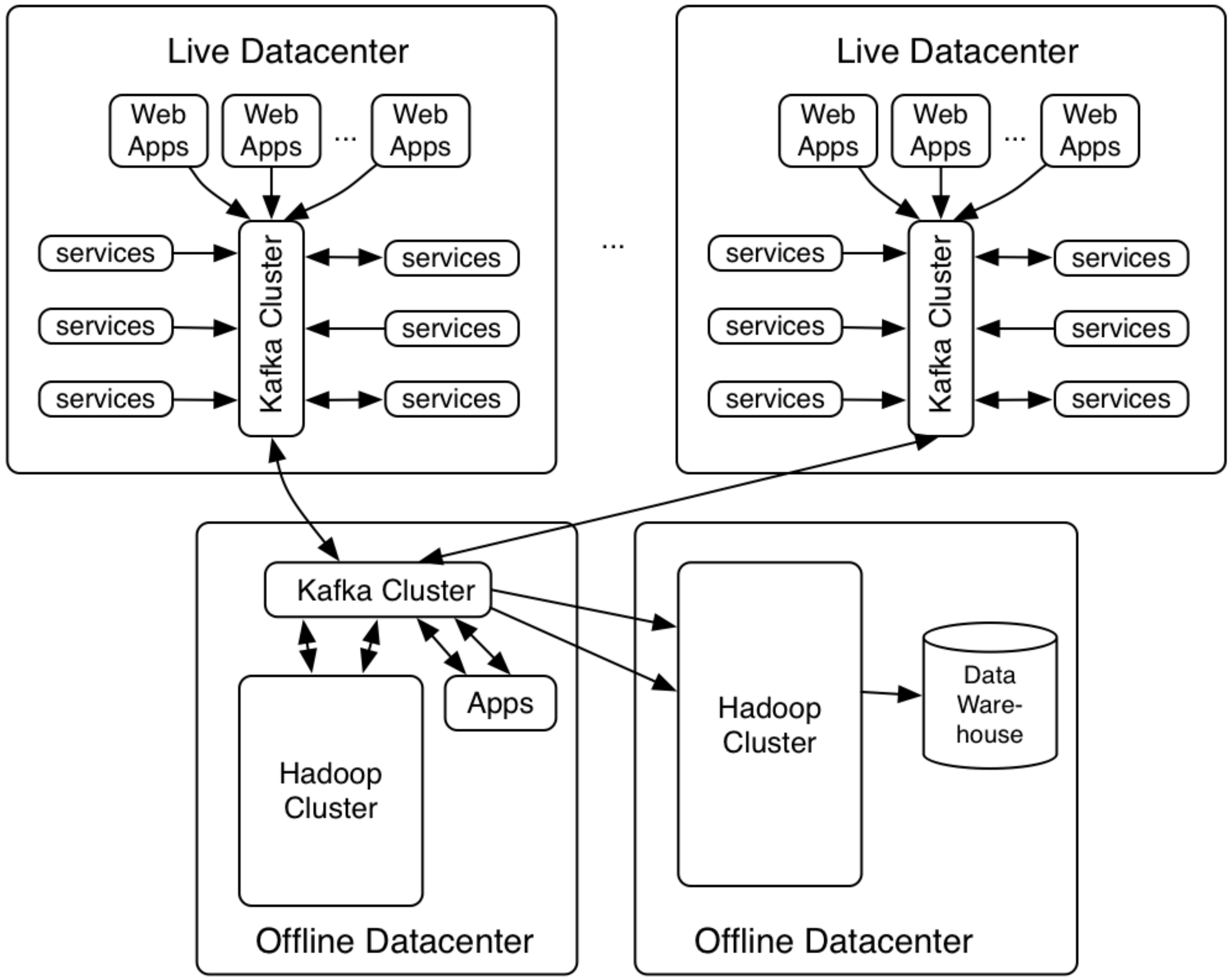
Zookeeper

- Highly available
- Manages shared state of any kind
- Transaction control
- Lock management
- Distributed hierarchical key-value store
- Written in Java, API in Java and C
- In memory
- Minimum of three nodes for production
- All nodes participate (ensemble, quorum)

Kafka Failover vs. Kafka Disaster Recovery

- Kafka uses replication for failover. Replication of Kafka topic log partitions allows for failure of a rack or AWS availability zone (AZ).
- You need a replication factor of at least 3 to survive a single AZ failure. You need to use Mirror Maker, a Kafka utility that ships with Kafka core, for disaster recovery.
- Mirror Maker replicates a Kafka cluster to another data-center or AWS region. They call what Mirror Maker does mirroring as not to be confused with replication.

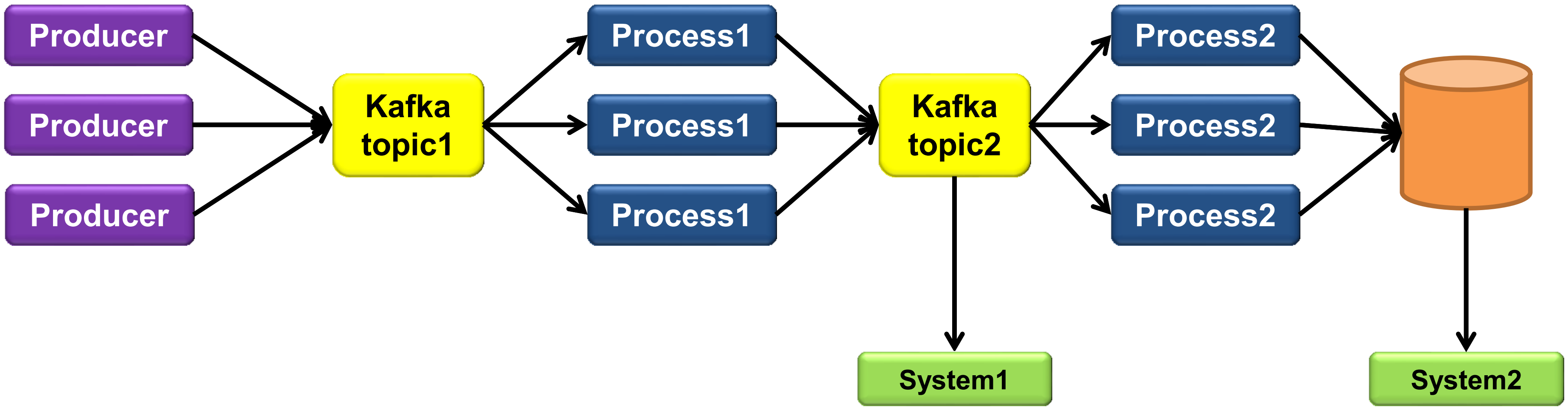
Multi-datacenter replication



Source: Kafka

Stream processing

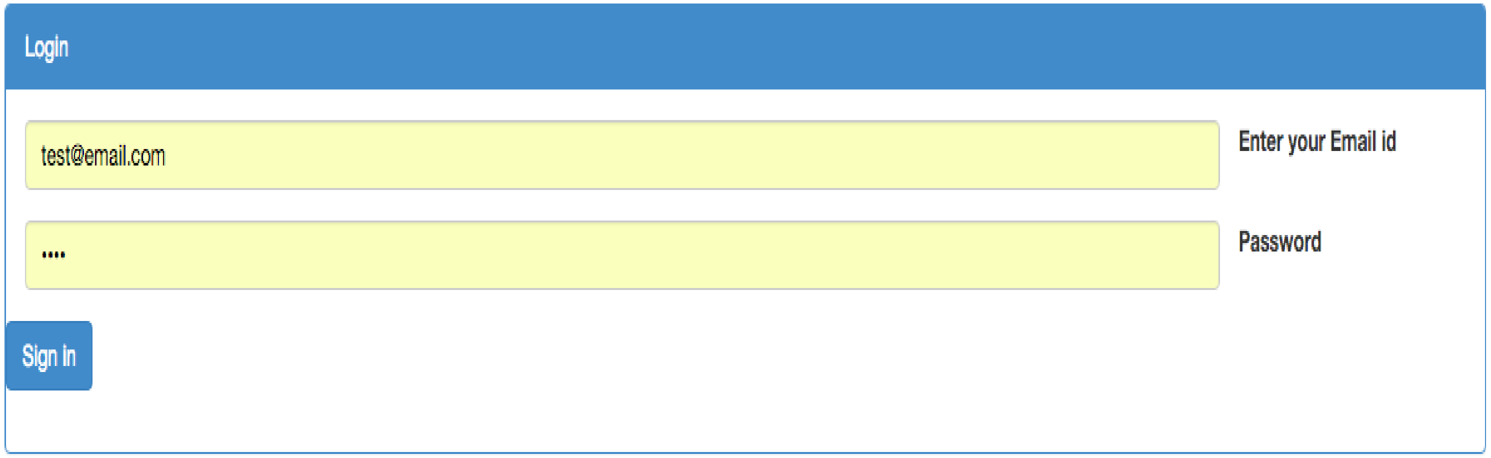
Kafka as a processing pipeline backbone



Example

- Login Application
- Create a server application and a client node.js application
- Create Kafka messaging queues to pass the data between client and server

User Interface



Enter your Email id: test@email.com

Password: ••••