



Apache Kafka from Scratch

©2013 LinkedIn Corporation. All Rights Reserved.

ORGANIZATION NAME



Todd Palino

Staff Site Reliability Engineer
Data Infrastructure Streaming

©2013 LinkedIn Corporation. All Rights Reserved.

ORGANIZATION NAME

Agenda

- Apache Zookeeper
- Apache Kafka
- Producing and Consuming Messages
- Kafka Clusters
- Use Cases for Kafka
- More Time? More Fun!



Schedule

- Tutorial, Part the First
 - 9:00 AM to 10:30 AM
- Break
 - 10:30 AM to 11:00 AM
- Tutorial, the Thrilling Conclusion
 - 11:00 AM to 12:30 PM



Before We Start

- Make sure you have Java 8 installed
 - Java 7 will work for now
 - Java 6 may be OK, but I won't guarantee it
- Also make sure you have the JAVA_HOME environment variable set properly in your .profile (or local variant)
- You will not need root privileges
 - We'll be using your home directory, and high numbered TCP ports



Who Do You Think You Are?

- Staff Site Reliability Engineer for Kafka at LinkedIn
- LinkedIn pushes a lot of data around
 - Over 1 trillion messages a day written to Kafka
 - Approaching 1 petabyte a day read out of Kafka
- We run a lot of Kafka
 - Over 75 clusters, in 10 different fabrics
 - Over 1300 Kafka brokers, each with at least 6.5 TB of useable storage



Apache Zookeeper



©2013 LinkedIn Corporation. All Rights Reserved.

ORGANIZATION NAME 7

Apache Zookeeper

- Distributed coordination service
- What does it have?
 - Hierarchical Namespace
 - Ordered Transactions
 - Ephemeral Nodes
 - Watches



©2013 LinkedIn Corporation. All Rights Reserved.

ORGANIZATION NAME 8

What is Zookeeper NOT?

- Queue
 - Yes, some people say you can do this
 - Not tuned for this kind of use, and it can overwhelm ZK
- Logging Service
 - Sequential nodes are provided as part of Zookeeper
 - Storing large amounts of data is not recommended
- Large Data Store
 - Zookeeper nodes have a size limit (default 1 MB)



Installing Zookeeper

- Unpack the zookeeper-3.4.6.tar.gz distribution
- Create a configuration file:

```
# cat > zookeeper-3.4.6/conf/zoo.cfg << EOF
> tickTime=2000
> dataDir=/tmp/zookeeper
> clientPort=2181
> EOF
```
- Create the /tmp/zookeeper directory
- Start Zookeeper:

```
zookeeper-3.4.6/bin/zkServer.sh start
```



Four Letter Words

- Zookeeper has basic admin functions on the port it is listening to
 - The commands are called “four letter words”
- Connect to port 2181 on localhost and try the following commands:
 - ruok
 - srvr
 - mntr
 - conf
 - cons



Zookeeper CLI

- Zookeeper ships with a basic CLI tool for viewing and modifying data
- Run it with the following command:
 - # `zookeeper-3.4.6/bin/zkCli.sh -server localhost:2181`
- Through this interface you can inspect the Zookeeper data
- You can also create, modify, and delete data
 - This means you can cause a massive amount of damage



Playing with Zookeeper

- Create a new node in the CLI:
`[zk: localhost:2181(CONNECTED) 0] create /foo bar`
- Get the information about that node:
`[zk: localhost:2181(CONNECTED) 1] get /foo`
- Update the data in that node:
`[zk: localhost:2181(CONNECTED) 2] set /foo baz`
`[zk: localhost:2181(CONNECTED) 3] get /foo`
- Delete the node:
`[zk: localhost:2181(CONNECTED) 4] delete /foo`



More About Zookeeper

- Zookeeper has more functionality than the basic CRUD operations
 - Watches
 - ACLs
 - rmr
- For getting started with Kafka, this is all we need to know
- Once you have Zookeeper running, you will want to use it for other things
 - Be smart: not everything is a nail to hit with your hammer



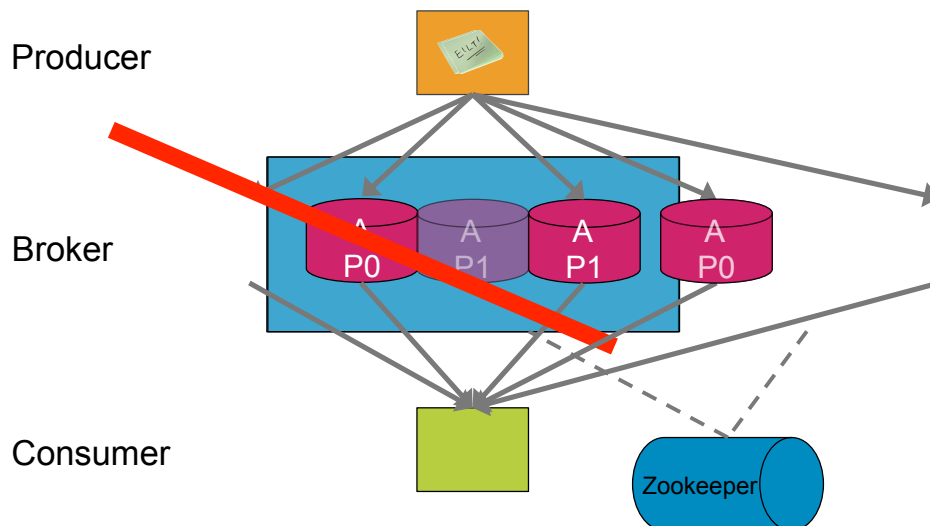
Apache Kafka



©2013 LinkedIn Corporation. All Rights Reserved.

ORGANIZATION NAME 15

Kafka Pub/Sub Basics



©2013 LinkedIn Corporation. All Rights Reserved.

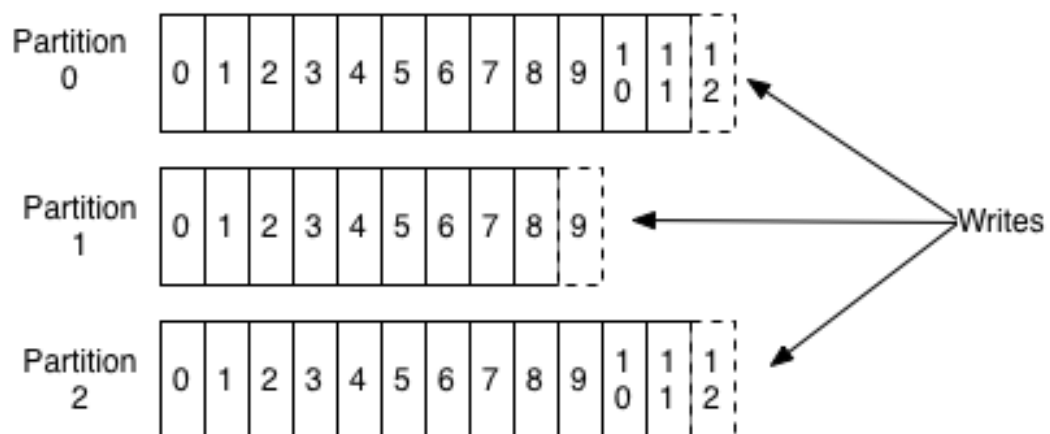
ORGANIZATION NAME 16

What Is Kafka?

- Publish / Subscribe messaging platform
 - Implemented as a distributed commit log
- Features
 - Disk Based
 - Durable
 - Scalable
 - Low Latency
 - Finite Retention



Distributed Commit Log



Installing Kafka

- Unpack the kafka_2.11-0.8.2.2.tgz distribution
- Create the /tmp/kafka-logs directory
- Start Kafka

```
kafka_2.11-0.8.2.2/bin/kafka-server-start.sh -daemon  
/usr/local/kafka/config/server.properties
```



Producing and Consuming



Message Schemas

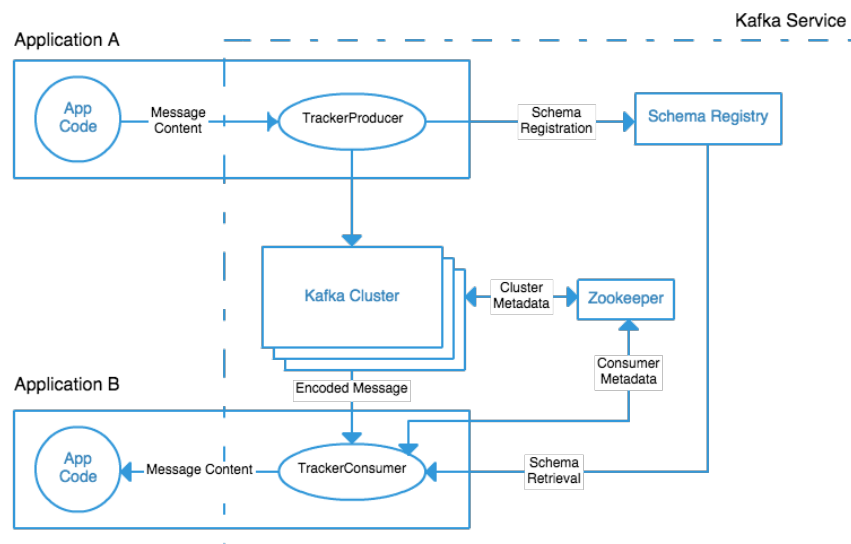
- Why enforce schemas?
 - Documentation for message formats
 - Decouple consumers from producers
- Types of schemas
 - JSON
 - Protobuf
 - Avro
- Centralized Registry
 - Not part of Kafka, but a good idea



©2013 LinkedIn Corporation. All Rights Reserved.

ORGANIZATION NAME 21

Kafka with Schema Registry



©2013 LinkedIn Corporation. All Rights Reserved.

ORGANIZATION NAME 22

Console Producer

- Simple CLI producer provided with Kafka
- Important options
 - `--broker-list` – semicolon separated list of host:port pairs
 - `--compress` – enable compression of messages
 - `--request-required-acks` – Set to 1 to wait for acknowledgement of receipt
 - `--topic` – Where to send your message
- More options exist to facilitate batching, serialization



Explicitly Creating a Topic

- Example assumes you have started Kafka on the default port of 9092, and your current working directory is the installation directory
- Execute the following command:

```
bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic  
mytopic --replication-factor 1 --partitions 1
```
- `replication-factor` and `partitions` are required options
 - `replication-factor` will be explained later
 - `partitions` is the number of partitions to create for the topic



Produce Messages to Kafka

- Start the kafka-console-producer.sh tool:

```
kafka_2.11-0.8.2.2/bin/kafka-console-producer.sh --broker-list  
localhost:9092 --topic mytopic
```
- Produce some messages
 - Messages are simple strings
 - Hitting “Enter” sends the message
 - Control-D sends EOF and closes the producer



Console Consumer

- CLI tool for consuming messages, provided with Kafka
 - Very useful for examining messages while debugging applications
 - Can also be used as a consumer client inside other apps
- Important options
 - `--zookeeper` – semicolon separated list of host:port[/path] strings for ZK servers
 - `--topic` – Where to consume messages from
 - `--from-beginning` – Specifies to start from the earliest offset
- Group name is selected to be unique
 - If you need to specify a group, add a properties file with `--consumer.config`



Consume Messages from Kafka

- Start the kafka-console-consumer.sh tool:

```
kafka_2.11-0.8.2.2/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic mytopic --from-beginning
```
- Note that the from-beginning flag is important
 - Otherwise, we would only get messages produced from now forward
- The messages that you produced previously will be displayed
- Experiment
 - Produce more messages while you consume
 - Start up another consumer and run it in parallel



Client Libraries

- Java libraries are the only ones supported by Kafka
 - Maintained as part of the Kafka code base
 - Provide first-class services for producers and consumers
- 3rd Party Clients
 - Exist for many languages, including C++, Go, Python, Perl, etc.
 - <https://cwiki.apache.org/confluence/display/KAFKA/Clients>
- Many do not provide balanced consumer implementations
 - Requires a significant investment in Zookeeper logic
 - One client's implementation may not work with another's



Options for 3rd Party Consumers

- Use a client with balanced consumer support
 - Go: https://github.com/stealthly/go_kafka_client
 - Python: <https://github.com/Parsely/pykafka>
 - C++: <https://github.com/edenhill/librdkafka>
- Use the console consumer
- Use an HTTP interface layer
- Wait for the new consumer interface



Producing Messages from Python

- Run the python-producer.py script
- Experiment
 - What happens if you change the topic name to an unknown topic?
 - What happens the next time you run it?



Consume Messages from Python

- Run the `python-consumer.py` script
- Note that this is a simple consumer that does not balance
 - The group name is used for storing offsets in Kafka
- Experiment
 - What happens when you consume from a topic that doesn't exist?
 - Modify the producer to continually send messages and run in parallel



Kafka Clusters



Why Cluster?

- Replication of messages
 - Multiple brokers can be replicas
 - Only one broker is the leader
- Horizontal scalability
 - Partitions for one topic can be distributed to many brokers
 - Good for networking, good for storage
- Redundant array of inexpensive brokers



Create a Second Broker

- Create a second data directory (e.g. /tmp/kafka-logs-2)
- Make a copy of the config/server.properties file
- Edit the new configuration file
 - Change the broker.id value to 1
 - Change the log.dirs config to the new data directory
- Start a second copy of Kafka with the new config file



Increasing Replication Factor for a Topic

- There's no simple way to do this
 - The kafka-topics command does not let you change the RF
 - We have to use partition reassignment to add another replica
- Create a file on disk named /tmp/reassign.json with the following content:

```
{"partitions": [{"topic": "mytopic", "partition": 0, "replicas": [0,1]}], "version":1}
```
- Execute the following command:

```
bin/kafka-reassign-partitions.sh -zookeeper localhost:2181 --reassignment-json-file /tmp/reassign.json --execute
```



Adding Partitions to a Topic

- Partitions can only be increased
- The cluster does not balance partitions to all brokers
 - Over time you will get hot nodes
 - External tools can be used for rebalancing the cluster
- Execute the following command:

```
bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic mytopic --partitions 2
```



Shutting Down a Broker

- With a replicated topic, there is now redundancy
- Kill one of the broker processes
- Run the console consumer with the `--from-beginning` flag to see that it still works



Use Cases



What Can I Use Kafka For?

- Simple Queuing
- Tracking User Events
- Log Aggregation
- Real-time Metrics



Use Case - Queuing

- Most common case people think of
- Example – Sending emails
 - You need to send emails from multiple applications
 - All emails should look the same, so they are decorated by a common method
- Why Kafka?
 - Multiple producers of emails send emails to Kafka
 - Decorator service consumes messages to send, scaling horizontally by partitions
 - Reduces the need to maintain decoration everywhere



Use Case - Tracking

- Collection of events, often with offline processing in Hadoop
- Example – Page Views
 - Users hit multiple pages, and multiple applications
 - Reports need to be generated in Hadoop on a many different factors
- Why Kafka?
 - Multiple applications can produce messages with ease
 - Messages are protected with replication while in transit
 - Retain messages in Kafka long enough to get them to Hadoop



Use Case - Logging

- Application logs need to be collected for multiple types of analysis
- Example
 - Logs need to be collected from several applications
 - Many people are interested in them – developers, security, operations
- Why Kafka?
 - Multiple producers are no problem
 - Each interested party can consume the logs without interfering with the others
 - Collection is separated from use



Use Case - Metrics

- You want to be able to watch application and system metrics in real time when debugging problems
- Example
 - Applications emit metrics to a topic with fine granularity
 - You can start up a tool at any point to consume specific metrics and display them
- Why Kafka?
 - Each user can have a randomized consumer group
 - Metrics can be retained in Kafka for a very short period of time to reduce storage
 - You can also consume metrics with lower granularity into permanent storage



Additional Topics



Message Retention

- Kafka retains and expires messages via three options
 - Time-based (the default, which keeps messages for at least 168 hours)
 - Size-based (configurable amount of messages per-partition)
 - Key-based (one message is retained for each discrete key)
- Time and size retention can work together, but not with key-based
 - With time and size configured, messages are retained either until the size limit is reached OR the time limit is reached, whichever comes first
- Retention can be overridden per-topic
 - Use the kafka-topics.sh CLI to set these configs



Balancing Partitions

- Kafka attempts to spread out partitions for a topic as they are created
 - It doesn't remember state from topic to topic, however
 - It also does not know how to add or remove a broker
- External scripts can be created to perform functions like this
- kafka-assigner.py is what we use at LinkedIn
 - It can add and remove brokers
 - It can also perform rebalances by partition count and size
 - Requires Python, as well as the Kazoo and Paramiko libraries



Going to Production

- Zookeeper Setup
 - Zookeeper stays running if you have $(n/2)+1$ nodes still talking to each other
 - Run 5 nodes, not 3 nodes
- Kafka Clusters
 - Run at least as many Kafka brokers in a cluster as you want replicas, plus one
 - For most applications, replication factor 2 is sufficient
 - For “no data loss”, replication factor of 3 or more is required
 - The RF will determine the number of broker failures you can suffer
- Spread out to multiple racks (or the equivalent)



Resources



More Talks

- ApacheCon 2014: Kafka as a Service
 - Presentation – <http://www.slideshare.net/ToddPalino/enterprise-kafka-kafka-as-a-service>
 - YouTube – <https://www.youtube.com/watch?v=7dkSze52i-o>
- ApacheCon 2015: Kafka at Scale: Multi-Tier Architectures
 - Presentation – <http://www.slideshare.net/ToddPalino/kafka-at-scale-multitier-architectures>
- Tuning Kafka for Fun and Profit
 - <http://www.slideshare.net/ToddPalino/tuning-kafka-for-fun-and-profit>



Getting Involved With Kafka

- <http://kafka.apache.org>
- Join the mailing lists
 - users@kafka.apache.org
 - dev@kafka.apache.org
- irc.freenode.net - #apache-kafka
- Meetups
 - Apache Kafka - <http://www.meetup.com/http-kafka-apache-org>
 - Bay Area Samza - <http://www.meetup.com/Bay-Area-Samza-Meetup/>
- Contribute code

