Kafka Administration and Integrations

Kafka Integration with Apache Storm

- Integration between Apache Kafka and Apache Storm is a popular and natural integration

- Let's take a whirlwind tour of Apache Storm and what it provides

Apache Storm

- **Apache Storm** - Distributed and fault-tolerant realtime computation
- Allows reliable processing of unbounded streams of data
- Developed by BackType initially and then acquired by Twitter and open-sourced
- Comparable to other streaming engines such as Spark Streaming and Flink

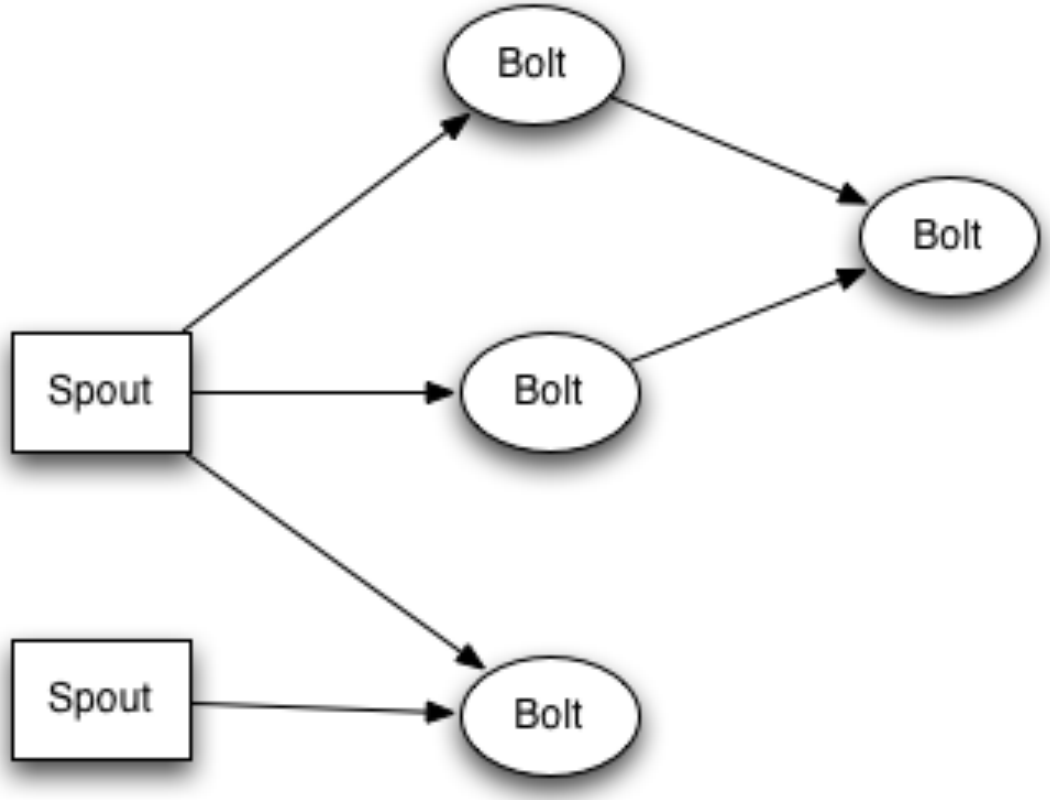
Major Storm Concepts

- **Topology**
  - Logic for an application that is analogous to a MapReduce job except that a topology runs forever or until killed
- **Streams**
  - An unbounded sequence of tuples
  - Storm provides a way to transform incoming data in a distributed and reliable way
  - For example, you may transform a stream of tweets into a stream of trending topics
- **Spouts**
  - A source of streams in a topology (e.g. the Twitter API)
  - Spouts read tuples from an external source and emit them into the topology

Major Storm Concepts

- **Bolts**
  - All processing in topologies is done in bolts
  - They are responsible for tasks such as filtering, aggregations, joins, talking to databases, etc
  - Complex stream transformations can require multiple steps and thus multiple bolts

A Storm Topology



Kafka and Storm Integration

- Storm has a built-in module called `storm-kafka-client` which is compatible with Kafka 0.10+
  - Legacy version of this module for Kafka 0.8 is called `storm-kafka`

- `storm-kafka-client` has a Storm *spout* implementation to consume data from Kafka using the *consumer* API

- It would also be possible of course to write a Storm *bolt* to *produce* data to Kafka although this is not as common

A Sample Data Flow

Kafka Integration with Hadoop

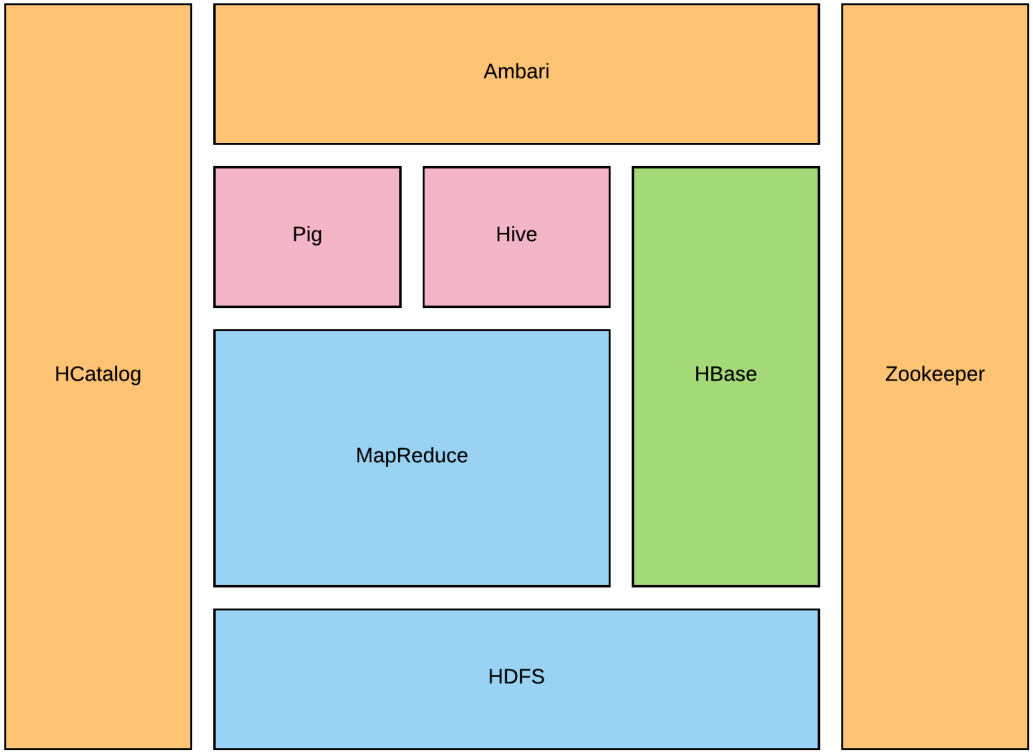
- Another common use case is consuming Kafka data directly into **Hadoop**
- **Hadoop** is a large scale distributed batch processing framework which parallelizes data processing across many servers
- Its strength is in the ability to scale across thousands of commodity servers that don't share memory or disk space

Major Functional Components

- **Hadoop** can be thought of as an ecosystem comprised of many different components that work together to create a single platform

- Two key functional components:
  - **HDFS** – a scalable file system that distributes and stores data across all machines in Hadoop cluster
  - **MapReduce** – the system used to efficiently process the large amount of data stored in HDFS. Large data is divided into smaller and smaller tasks in a pipeline.

Hadoop Cluster Components



Kafka Integration with Hadoop

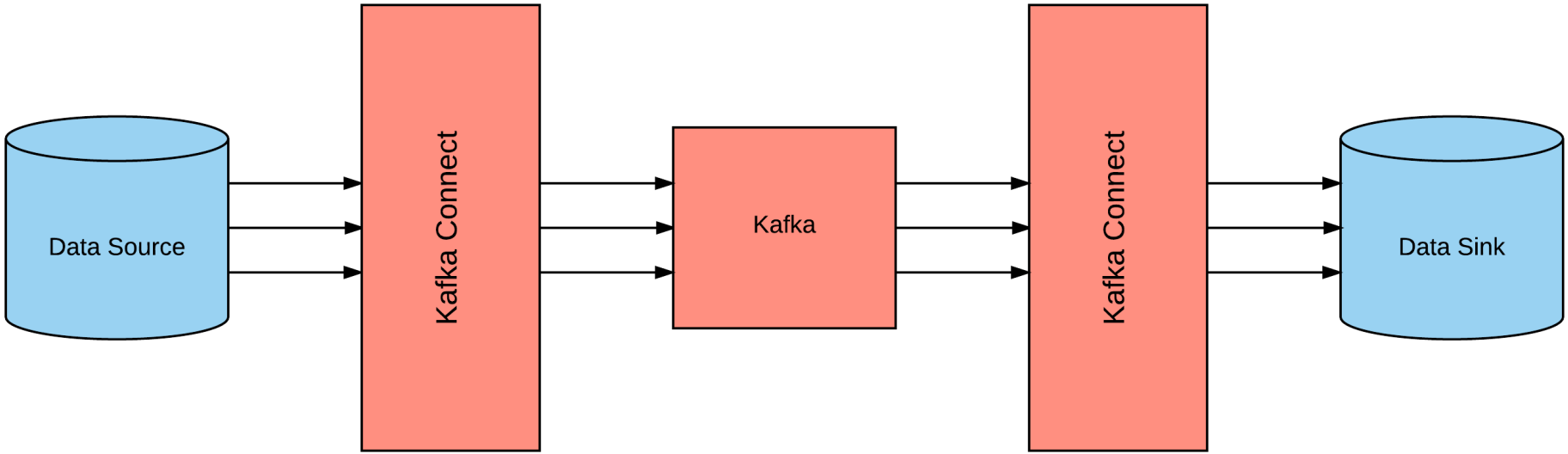
- In earlier versions of Kafka (0.8), integration with Hadoop could be achieved with an extension directly in the Kafka code base

- This was great news for Hadoop users but a lot of code would need to be written for other integrations
  - S3
  - HBase
  - JDBC
  - Cassandra

Kafka Connect and Confluent

- **Confluent** is a startup that was founded in 2014 by the creators of Kafka
- **Kafka Connect** is a feature in Apache Kafka 0.9+ developed by **Confluent** that makes building and managing stream data pipelines easier
- **Kafka Connect** abstracts away common problems that every connector to Kafka needs to solve such as
  - fault tolerance
  - partitioning
  - offset management
  - monitoring

High Level Look at Kafka Connect



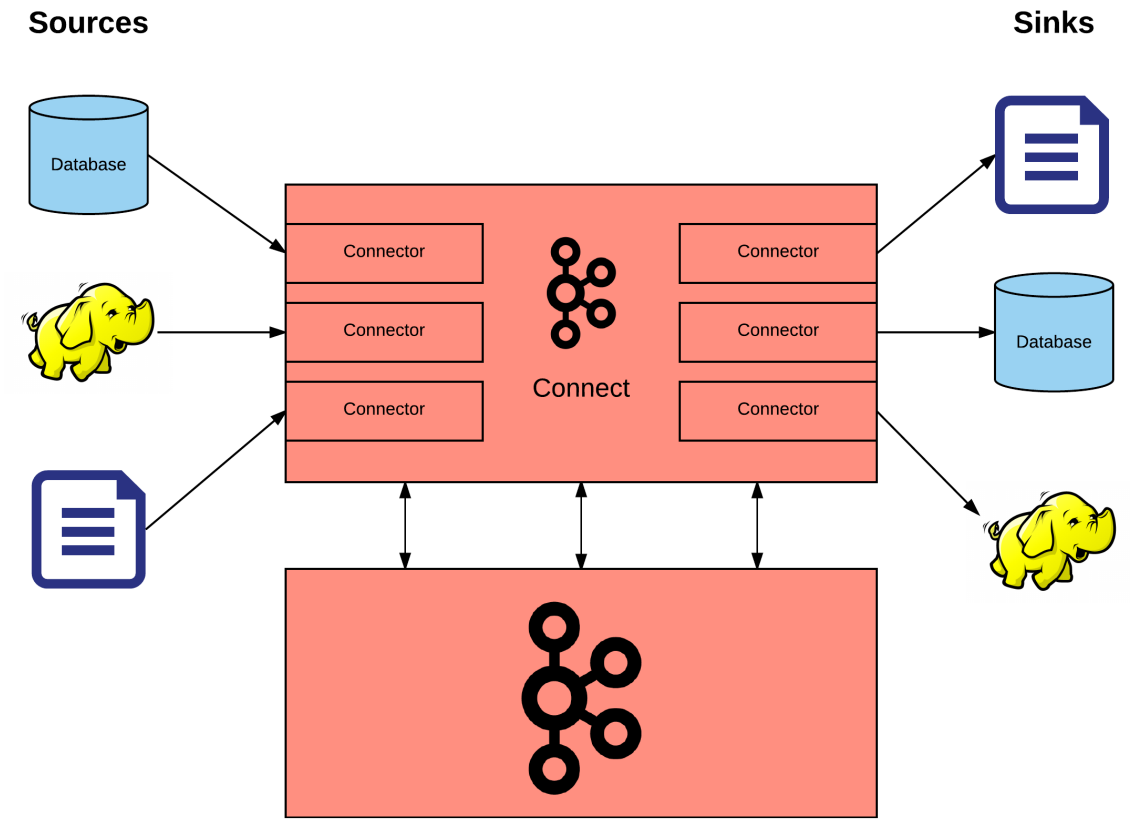
Certified Connectors

- Confluent maintains the **HDFS (Sink) connector** using the Kafka Connect framework
- Other certified connectors from Confluent include
  - JDBC (Source)
  - Elastic Search (Sink)
  - Attunity (Source)
  - Couchbase (Source)
  - GoldenGate (Source)
  - JustOne (Sink)
  - Syncsort DMX (Source)
  - Syncsort DMX (Sink)
  - Vertica (Source)
  - Vertica (Sink)

Additional Community Connectors

- Other notable community connectors utilizing Kafka Connect
  - Cassandra (Source and Sink)
  - Elastic Search (Sink)
  - FTP (Source)
  - HBase (Sink)
  - InfluxDB (Sink)
  - Bloomberg Ticker (Source)
  - MongoDB (Source)
  - Mixpanel (Source)
  - MQTT (Source)
  - Solr (Sink and Source)
  - And more!

Another Kafka Connect View



Kafka Administration

- There are a number of useful operations that are not automated and must be triggered using one of the tools that ship with Kafka
  - Adding/deleting topics
  - Modifying topics
  - Graceful shutdown
  - Balancing leadership
  - Checking consumer position
  - Mirroring data between clusters
  - Expanding a cluster
  - Decommissioning brokers
  - Increasing replication factor

Adding and Deleting Topics

- Topics can be added manually or you can have them be created automatically when data is first published to a non-existent topic

- Topics are added and removed with the topic tool
  - `kafka-topics.sh --zookeeper zk_host:port --create --topic my_topic_name --partitions 20 --replication-factor 3 --config x=y`
  - `kafka-topics.sh --zookeeper zk_host:port --delete --topic my_topic_name`

Modifying Topics

- Changing the configuration or partitioning of a topics can be done with the same topic tool
  - kafka-topics.sh --zookeeper zk_host:port/chroot --alter --topic my_topic_name --partitions 40

- The partition count controls how many logs the topic will be sharded into

Graceful Shutdown

- A Kafka cluster will automatically detect any broker shutdown or failure and elect new leaders for the partitions on that machine
- When a broker is brought down intentionally for configuration changes, Kafka supports a more graceful shutdown that comes with two optimizations
  - All logs are synced to disk to avoid needing to validate checksums when restarting. This speeds up intentional restarts
  - Partitions the server is the leader for will be migrated to other replicas prior to shutting down. This will make leadership transfer faster and minimize the time each partition is unavailable to a few milliseconds

Graceful Shutdown Continued

- To take advantage of this graceful shutdown, the following setting is required
  - `controlled.shutdown.enable=true`

Checking Consumer Position

- The following command will show the position of all consumers in a consumer group and how far behind the end of the log they are

  - `kafka-run-class.sh kafka.tools.ConsumerOffsetChecker --zookeeper localhost:2181 --group test`

Mirroring Data Between Clusters

- Replicating data *between* Kafka clusters is known as **mirroring** which is not to be confused with the replication that happens amongst the nodes in single cluster

- Kafka comes with a tool for this purpose called `kafka-mirror-maker`

- A common use case is to provide a replica in another datacenter

Expanding a Cluster

- To add servers to a Kafka cluster, just assign them a unique broker id and start
- The new servers will not automatically be assigned any data partitions so unless partitions are moved to them they won't be doing any work until new topics are created
- Most of the time when you add machines to the cluster you will migrate some existing data with the `kafka-reassign-partitions` tool
- Check out the documentation for examples and usage!

Decommissioning Brokers

- The partition reassignment tool cannot automatically generate a reassignment plan for decommissioning brokers
- The admin must come up with a plan to move the replica for all partitions hosted on the broker to be decommissioned to the rest of the brokers
  - Can be **tedious**!
- Tooling support for this task is planned in the future

Increasing Replication Factor

- To increase the replication factor of an existing partition, the extra replicas must be specified in a JSON file

- The following example increases the replication factor of partition 0 of topic `test` from 1 to 3
  - increase-replication-factor.json:

```
{"version":1,"partitions":[{"topic":"test","partition":0,
"replicas":[5,6,7]}]}
```

  - `kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file increase-replication-factor.json --execute`

Monitoring

- Kafka uses **Yammer Metrics** for metrics reporting in both the server and client

- You can use `jconsole` and attach to a running Kafka clinet or server to see all the metrics available with JMX

Security

- In release 0.9, Kafka added a number of security related features to Kafka
  - Authentication of connection to brokers from clients using SSL or SASL (Kerberos)
  - Authentication of connections from brokers to Zookeeper
  - Encryption of data transferred between brokers and clients, between brokers, or between brokers and tools using SSL
  - Authorization of read/write operations by clients
  - Authorization is pluggable and can be integrated with external authorization services

Summary

- Kafka Integrations
  - Apache Storm
  - Apache Hadoop
- Kafka Connect
- Administration
  - Command line tools for modifying topics, graceful shutdown, mirroring data, checking consumer position, etc.
  - Monitoring
  - Security