Kafka Ecosystem

Kafka - Brokers - Topics - Partitions - ISR - Writing and Reading Data from Kafka

Realiability

Production Reliability

- How many in-sync replicas require to have acked. #### Consumption Reliability
- guarantees the data is read at least **once**
- data quality

Kafka Connect

- Data centric pipeline: Uses meaningful data abstraction to pull or push data from/into Kafka

Kafka Connect Source

- e.g. Get data from a data source to insert into a Kafka Topic ### Kafka Connect Sink

Event Consumption

Custom Consumer

Current Consumer

1. Consume a single message from a topic;
2. Using the header, determine the avro schema that has to deserialize the message;
3. Deserialize message;
4. Send signals to the interested functions;
5. Commit the message offset.

Issues

1. Does not allow for batch consumption;
   - This could be done using the consume() method instead of the poll().
2. The avro schemas currently used with the running version of fastavro, is no longer compatible with the most recent version of the same library (We are stuck with the same version of fastavro);
3. No single source of truth for the avro schemas;
4. When updating a schema's version, for the information to be

Kubernetes and Scalability

Kubernetes as a Service

- Kubernetes runs as a distributed cluster, providing us with an optimal solution for horizontal scalability.
- Most of the time, we do not require more computing power, as most of the processing time within a service, has to do with network availability.

Internal Scalability

- By default, Kubernetes allows for an instance to scale based on metrics like CPU and memory, allowing for a service to be available regardless of the load. With exception to the number of pods having reached the limit of instances, or the machines within the Kubernetes cluster, no longer having capacity to run anymore instances.

Custom and External Metrics

Since Kubernetes version 1.6. it provided a new autoscaling API, allowing for other metrics to be taken into consideration when scaling objects.

Proposal

This has been a build up to the following solution, which we found is adequate to solve the problems that were presented to our team.

As context, these are the requirements for our solution: - Reliability - Near Real-time - Monitoring

The remaining of the presentation will be to present with an architecture that will respond to each of these requirements.

Architecture

[[image]]

Dependencies

- Kubernetes, or any service that provides distributed scaling.

Types of Consumers

Any of the consumers previously mentioned: Connect; DeserializingConsumer; our CurrentConsumer, can be used within our solution. But, to achieve a near real-time data consumption, the way a topic is partitioned, will influence how many consumer we can scale up to, limiting our scalability to the amount of partitions we