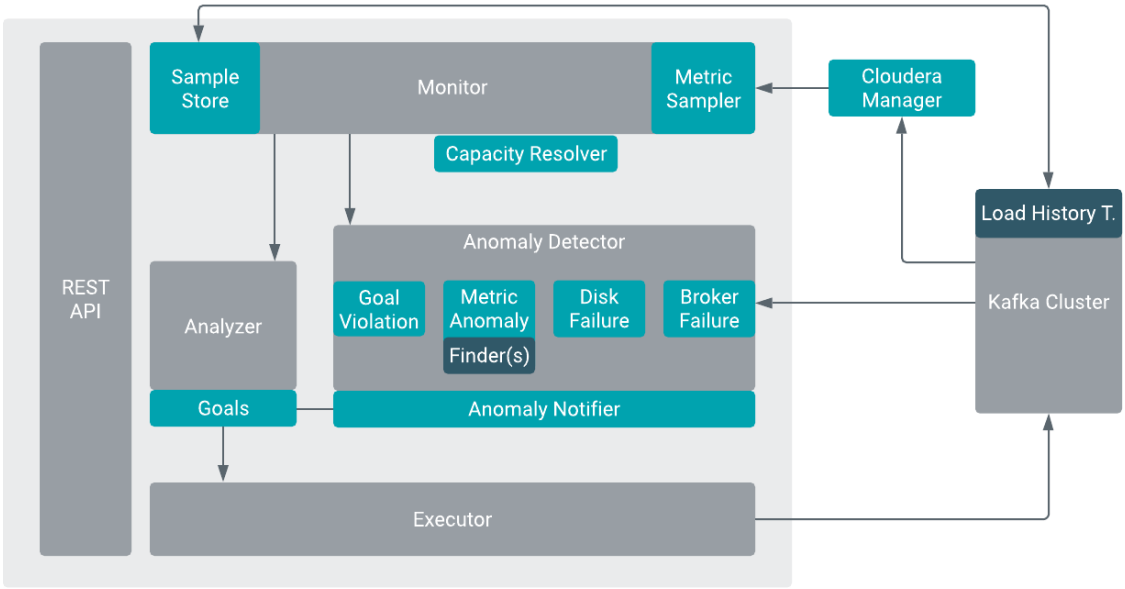
Kafka cluster load balancing using Cruise Control

You can use Cruise Control as a load balancing component in large Kafka installations to automatically balance the partitions based on specific conditions for your deployment. The elements in the Cruise Control architecture are responsible for different parts of the rebalancing process that uses Kafka metrics and optimization goals.

The following illustration shows the architecture of Cruise Control.



Load Monitor

Generates a cluster workload model based on standard Kafka metrics and resource metrics to utilize disk, CPU, bytes-in rate and bytes-out rate. Feeds the cluster model into Anomaly Detector and Analyzer.

Analyzer

Generates optimization proposals based on optimization goals provided by the user, and cluster workload model from Load Monitor. Hard goals and soft goals can be set. Hard goals must be fulfilled, while soft goals can be left unfulfilled if hard goals are reached. The optimization fails if the hard goal is violated by optimization results.

Anomaly Detector

Responsible for detecting anomalies that can happen during the rebalancing process. The following anomaly detections are supported in Cruise Control:

- Broker failure
- Goal violations
- Disk failure
- Slow broker as Metric Anomaly
- Topic replication factor

The detected anomalies can be resolved by the self-healing feature of Cruise Control. For more information, see the How Cruise Control self-healing works documentation.

Executor

Carries out the optimization proposals and it can be safely interrupted when executing proposals. The executions are always resource-aware processes.

How Cruise Control retrieves metrics

Cruise Control creates metric samples using the retrieved raw metrics from Kafka. The metric samples are used to set up the cluster workload model for the Load Monitor. When deploying Cruise Control in a CDP environment, Cloudera Manager executes the process of retrieving the metrics from Kafka to Cruise Control.

In Load Monitor, the Metric Fetcher Manager is responsible for coordinating all the sampling tasks: the Metric Sampling Task, the Bootstrap Task and the Linear Model Training Task.

Each sampling task is carried out by a configured number of Metric Fetcher threads. Each Metric Fetcher thread uses a pluggable Metric Sampler to fetch samples. Each Metric Fetcher is assigned with a few partitions in the cluster to get the samples. The metric samples are organized by the Metric Sample Aggregator that puts each metric sample into a workload snapshot according to the timestamp of a metric sample.

The cluster workload model is the primary output of the Load Monitor. The cluster workload model reflects the current replica assignment of the cluster and provides interfaces to move partitions or replicas. These interfaces are used by the Analyzer to generate optimization solutions.

The Sample Store stores the metric and training samples for future use.

With the metric sampler, you can deploy Cruise Control to various environments and work with the existing metric system.

When you use Cruise Control in the Cloudera environment, HttpMetricsReporter reports metrics to the Cloudera Manager time-series database. As a result, the Kafka metrics can be read using Cloudera Manager.

How Cruise Control self-healing works

The Anomaly detector is responsible for the self-healing feature of Cruise Control. When self-healing is enabled in Cruise Control, the detected anomalies can trigger the attempt to automatically fix certain types of failure such as broker failure, disk failure, goal violations and other anomalies.

Broker failures

The anomaly is detected when a non-empty broker crashes or when a broker is removed from a cluster. This results in offline replicas and under-replicated partitions When the broker failure is detected, Cruise Control receives an internal notification. If self-healing for this anomaly type is enabled, Cruise Control will trigger an operation to move all the offline replicas to other healthy brokers in the cluster. Because brokers can be removed in general during cluster management, the anomaly detector provides a configurable period of time before the notifier is triggered and the self-healing process starts. If a broker disappears from a cluster at a timestamp specified as T, the detector will start a countdown. If the broker did not rejoin the cluster within the broker.failure.alert.threshold.ms since the specified T, the broker is defined as dead and an alert is triggered. Within the defined time of broker.failure.self.healing.threshold.ms, the self-healing process is activated and the failed broker is decommissioned.

Goal violations

The anomaly is detected when an optimization goal is violated. When an optimization goal is violated, Cruise Control receives an internal notification. If selfhealing for this anomaly type is enabled, Cruise Control will proactively attempt to address the goal violation by automatically analyzing the workload, and executing optimization proposals.

You need to configure the anomaly.detection.goals if you enable self-healing for goal violations to choose which type of goal violations should be considered. By default, the following goals are set for anomaly.detection.goals:

- com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal
- com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal
- com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal

Disk failure

The anomaly is detected when one of the non-empty disks dies. Note: This is the only anomaly that is related to Kafka broker running on a JBOD disk. When a disk fails, Cruise Control will send out a notification. If self-healing for this anomaly type is enabled, Cruise Control will trigger an operation to move all the offline replicas by replicating alive replicas to other healthy brokers in the cluster.

Slow broker

The anomaly is detected when based on the configured thresholds, a broker is identified as a slow broker. Within the Metric anomaly, you can set the threshold to identify slow brokers. In case a broker is identified as slow, the broker will be decommissioned or demoted based on the configuration you set for the remove.slow.broker. The remove.slow.broker configuration can be set to true, this means the slow broker will be removed. When setting the remove.slow.broker configuration to false, the slow broker will be demoted.

Topic replication factor

The anomaly is detected when a topic partition does not have the desired replication factor. The topic partition will be reconfigured to match the desired replication factor.