Cloudera Runtime 7.1.6

# Configuring Cruise Control

**Date published: 2020-05-04**
**Date modified: 2021-03-03**

**https://docs.cloudera.com/**

# Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

# Contents

**Adding Cruise Control as a service.....4**

**Setting capacity estimations and goals.....4**

Configuring capacity estimations.....4
Configuring goals.....5
Example of Cruise Control goal configuration.....7

**Enabling self-healing in Cruise Control.....7**

Changing the Anomaly Notifier Class value to self-healing.....7
Enabling self-healing for all or individual anomaly types.....8
Adding self-healing goals to Cruise Control in Cloudera Manager.....12

# Adding Cruise Control as a service

You need to use the Add Service wizard in Cloudera Manager to have Cruise Control service on your cluster. After selecting the host you want to add the Cruise Control role to, you need to review the default configurations. These configurations can also be set later after adding Cruise Control as a service.

### About this task

**Note:** It is recommended to add Cruise Control on the same CDP cluster as Kafka.

### Procedure

1. Open Cloudera Manager.
2. Select the drop-down menu to the right of your cluster on the Home screen.
3. Select Add Service.
   The Add Service wizard starts.
4. Select Cruise Control as the type of service from the list of services, then click Continue.
5. Assign roles to Cruise Control, then click Continue.
6. Check the default and suggested settings for configuration parameters on the Review Changes page, and set if needed.
7. Click Continue and wait until the first run of the Cruise Control service is completed.
8. Click Continue and then Finish.

### Results

You have added Cruise Control as a service in Cloudera Manager.

# Setting capacity estimations and goals

Cruise Control rebalancing works using capacity estimations and goals. You need to configure the capacity estimates based on your resources, and set the goals for Cruise Control to achieve the Kafka partition rebalancing that meets your requirements.

When configuring Cruise Control, you need to make sure that the Kafka topics and partitions, the capacity estimates, and the proper goals are provided so the rebalancing process works as expected.

You can find the capacity estimate and goal configurations at the following location in Cloudera Manager:

1. Go to your cluster in Cloudera Manager.
2. Select Cloudera Manager from the services.
3. Select Cruise Control from the list of Services.
4. Click Configuration.
5. Select Main from the Filters.

## Configuring capacity estimations

The values for capacity estimation needs to be provided based on your available resources for CPU and network. Beside the capacity estimation, you also need to provide information about the broker and partition metrics. You can set the capacity estimations and Kafka properties in Cloudera Manager.

For the rebalancing, you need to provide the capacity values of your resources. These values are used for specifying the rebalancing criteria for your deployment. The following capacity values must be set:

| Capacity | Description |
|---|---|
| capacity.default.cpu | 100 by default |
| capacity.default.network-in | Given by the internet provider |
| capacity.default.network-out | |

**Note:** For the capacity estimates, the disk capacity value is also needed. However, Cruise Control automatically retrieves the disk capacity value from the kafka_log_directory_disk_total_space Kafka metric.

The optimizers in Cruise Control use the network incoming and outgoing capacities to define a boundary for optimization. The capacity estimates are generated and read by Cruise Control. A capacity.json file is generated when Cruise Control is started. When a new broker is added, Cruise Control uses the default broker capacity values. However, in case disk related goals are used, Cruise Control must be restarted to load the actual disk capacity metrics of the new broker.

The following table lists all the configurations that are needed to configure Cruise Control specifically to your environment:

**Note:** The security settings are not listed in the table below.

| Configuration | Description |
|---|---|
| num.metric.fetchers | Parallel threads for fetching metrics from the Cloudera Manager database |
| partition.metric.sample.store.topic | Storing Cruise Control metrics |
| broker.metric.sample.store.topic | Storing Cruise Control metircs |
| partition.metrics.window.ms | Time window size for partition metrics |
| broker.metrics.window.ms | Time window size for broker metrics |
| num.partition.metrics.windows | Number of stored partition windows |
| num.broker.metrics.windows | Number of stored broker windows |

## Configuring goals

After setting the capacity estimates, you can specify which goals need to be used for the rebalancing process in Cloudera Manager. The provided goals are used for the optimization proposal of your Kafka cluster.

### Procedure

1. Access the **Configuration** page of Cruise Control.
   a) Go to your cluster in Cloudera Manager.
   b) Select Cruise Control from the list of services.
   c) Click Configuration.
2. Search for goals using the search bar.
   The list of goals are displayed based on the goal sets.

3. Add goals using the property name to the Default, Supported, Hard, Self-healing and Anomaly detection lists based on your requirements, and click Save Changes.

   The following table lists the goals that can be used:

| Goal | Property name | Description |
|---|---|---|
| `RackAwareDistributionGoal` | com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareDistributionGoal | As long as replicas of each partition can achieve a perfectly even distribution across the racks, this goal lets placement of multiple replicas of a partition into a single rack. |
| `ReplicaCapacityGoal` | com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal | Attempt to make all the brokers in a cluster to have less than a given number of replicas. |
| `CapacityGoals` | com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal | Goals that ensure the broker resource utilization is below a given threshold for the corresponding resource. |
| | com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal | |
| | com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal | |
| | com.linkedin.kafka.cruisecontrol.analyzer.goals.CpuCapacityGoal | |
| `ReplicaDistributionGoal` | com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaDistributionGoal | Attempt to make all the brokers in a cluster to have a similar number of replicas. |
| `PotentialNwOutGoal` | com.linkedin.kafka.cruisecontrol.analyzer.goals.PotentialNwOutGoal | A goal that ensures the potential network output (when all the replicas become leaders) on each of the brokers do not exceed the broker's network outbound bandwidth capacity. |
| `ResourceDistributionGoals` | com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskUsageDistributionGoal | Attempt to make the resource utilization variance among all the brokers are within a certain range. This goal does not do anything if the cluster is in a low utilization mode (when all the resource utilization of each broker is below a configured percentage.) This is not a single goal, but consists of the following separate goals for each of the resources. |
| | com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundUsageDistributionGoal | |
| | com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundUsageDistributionGoal | |
| | com.linkedin.kafka.cruisecontrol.analyzer.goals.CpuUsageDistributionGoal | |
| `TopicReplicaDistributionGoal` | com.linkedin.kafka.cruisecontrol.analyzer.goals.TopicReplicaDistributionGoal | Attempt to make the replicas of the same topic evenly distributed across the entire cluster. |
| `LeaderReplicaDistributionGoal` | com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderReplicaDistributionGoal | Attempt to make all the brokers in a cluster to have the similar number of leader replicas. |
| `LeaderBytesInDistributionGoal` | com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderBytesInDistributionGoal | Attempt to make the leader bytes in rate on each host to be balanced. |
| `PreferredLeaderElectionGoal` | com.linkedin.kafka.cruisecontrol.analyzer.goals.PreferredLeaderElectionGoal | Attempt to make the first replica in the replica list leader replica of the partition for all topic partitions. |
| `KafkaAssignerGoals`[1] | com.linkedin.kafka.cruisecontrol.analyzer.kafkaassigner.KafkaAssignerDiskUsageDistributionGoal | A goal that ensures all the replicas of each partition are assigned in a rack aware manner. |
| | com.linkedin.kafka.cruisecontrol.analyzer.kafkaassigner.KafkaAssignerEvenRackAwareGoal | Attempt to make all the brokers in a cluster to have the similar number of replicas |

[1] These goals are used to make Cruise Control behave like a Kafka assigner tool. These goals will be picked up if kafka_assigner parameter is set to true in the corresponding request (for example, with the rebalance request as shown in the Cruise Control documentation).

## Example of Cruise Control goal configuration

By default, Cruise Control is configured with a set of Default, Supported, Hard, Self-healing and Anomaly detection goals in Cloudera Manager. The default configurations can be changed based on what you would like to achieve with the rebalancing.

The following example details how to configure Cruise Control to achieve the following:

- Find dead/failed brokers and create an anomaly to remove load from them (self.healing.broker.failure.enabled)
- Move load back to the brokers when the brokers are available again (self.healing.goal.violation.enabled and added goals)
- Prevent too frequent rebalances to reduce cluster costs (incremented thresholds, reduced self.healing.goals set)
- Have an always balanced cluster from the replicas and leader replicas point of view
- Not enable every type of self-healing methods if it is not required (only two type of self-healing is enabled)

Configurations that need to be added to the Cruise Control Server Advanced Configuration Snippet (Safety Valve) for cruisecontrol.properties property:

- self.healing.goal.violation.enabled=true
- self.healing.broker.failure.enabled=true
- self.healing.exclude.recently.removed.brokers=false

Configurations that need to be set (and available explicitly among properties):

- anomaly.notifier.class=com.linkedin.kafka.cruisecontrol.detector.notifier.SelfHealingNotifier
- replica.count.balance.threshold=1.25
- leader.replica.count.balance.threshold=1.25

Goals that need to be added to Hard goals:

- com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaDistributionGoal
- com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderReplicaDistributionGoal

Goals that need to be added to Self-healing goals:

- com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaDistributionGoal
- com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderReplicaDistributionGoal

Goals that need to be added to Anomaly detection goals:

- com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaDistributionGoal
- com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderReplicaDistributionGoal

Other configurations can remain as set by default.

# Enabling self-healing in Cruise Control

To enable Cruise Control self-healing, you need to set the Anomaly Notifier Class, enable the self-healing for the anomaly types using the REST API and add self-healing goals in Cloudera Manager.

## Changing the Anomaly Notifier Class value to self-healing

You need to change the default anomaly notifier class to self-healing to enable the internal notification in Cruise Control that triggers the automatic self-healing.

### Procedure

1. Go to your cluster in Cloudera Manager.

2. Select Cruise Control from the list of Services.
3. Click on Configuration tab.
4. Search for the Cruise Control Server Advanced Configuration Snippet (Safety Valve) for cruisecontrol.properties setting.
5. Add the following parameter to the Safety Valve field:

```
anomaly.notifier.class=com.linkedin.kafka.cruisecontrol.detector.notifie
r.SelfHealingNotifier
```

6. Click Save changes.
7. Click on Action > Restart next to the Cruise Control service name to restart Cruise Control.

## Enabling self-healing for all or individual anomaly types

Self-healing is disabled for Cruise Control by default. You can enable self-healing in Cloudera Manager using the cruisecontrol.properties configuration, or with a curl POST request and the corresponding anomaly type.

### Enabling self-healing in Cloudera Manager

1. Go to your cluster in Cloudera Manager.
2. Select Cruise Control from the list of Services.
3. Click on Configuration tab.
4. Search for the Cruise Control Server Advanced Configuration Snippet (Safety Valve) for cruisecontrol.properties setting.
5. Choose to enable self-healing for all or only specific anomaly types, and add the corresponding parameter to the Safety Valve field based on your requirements.
   a. To enable self-healing for all anomaly types, add self.healing.enabled=true configuration parameter to the Safety Valve.
   b. To enable self-healing for specific anomaly types, add the corresponding configuration parameter to the Safety Valve:
      - self.healing.broker.failure.enabled=true
      - self.healing.goal.violation.enabled=true
      - self.healing.disk.failure.enabled=true
      - self.healing.topic.anomaly.enabled=true
      - self.healing.slow.broker.removal.enabled=true
      - self.healing.metric.anomaly.enabled=true
      - self.healing.maintenance.event.enabled=true
6. Provide additional configuration to self-healing.

   There are additional configurations that you can use to further customize the self-healing process.

| Configuration | Value | Description |
|---|---|---|
| anomaly.notifier.class | *com.linkedin.kafka.cruisecontrol.detector.notifier.SelfHealingNotifier* | The notifier class to trigger an alert when an anomaly is violated. The notifier class must be configured to enable self-healing. For more information, see Enabling self-healing in Cruise Control. |
| broker.failure.alert.threshold.ms | *900,000* | Defines the threshold to mark a broker as dead. If a non-empty broker leaves the cluster at time *T* and did not join the cluster before *T* + broker.failure.alert.threshold.ms, the broker is defined as dead broker since T. An alert will be triggered in this case. |

| Configuration | Value | Description |
|---|---|---|
| broker.failure.self.healing.threshold.ms | *1,800,000* | If self-healing is enabled and a broker is dead at T, self-healing will be triggered at T + broker.failure.self.healing.threshold.ms. |

For more information about the Self- healing configurations, see the Cruise Control documentation.

7. Provide additional configuration to the anomaly types.

   There are additional configurations that you can provide for the anomaly types.

| Anomaly type | Configuration | Value | Description |
|---|---|---|---|
| Broker failure | broker.failures.class | *com.linkedin.kafka.cruisecontrol.detector.BrokerFailures* | The broker failures class that extends broker failures. |
| | failed.brokers.file.path | *fileStore/failedBrokers.txt* | The file path to store the failed broker list. This is to persist the broker failure time in case Cruise Control failed and restarted when some brokers are down. |
| | fixable.failed.broker.count.threshold | *10* | The upper boundary of concurrently failed broker counts that are taken as fixable. If too many brokers are failing at the same time, it is often due to something more fundamental going wrong and removing replicas from failed brokers cannot alleviate the situation. |
| | fixable.failed.broker.percentage.threshold | *0.4* | The upper boundary of concurrently failed broker percentage that are taken as fixable. If a large portion of brokers are failing at the same time, it is often due to something more fundamental going wrong and removing replicas from failed brokers cannot alleviate the situation. |
| | broker.failure.detection.backoff.ms | *300000* | The backoff time in millisecond before broker failure detector triggers another broker failure detection if currently detected broker failure is not ready to fix. |
| | kafka.broker.failure.detection.enable | *false* | Whether to use the Kafka API to detect broker failures instead of ZooKeeper. When enabled, zookeeper.connect does not need to be set. |
| | broker.failure.detection.interval.ms | *null* | The interval in millisecond that broker failure detector will run to detect broker failures. If this interval time is not specified, the broker failure detector will run with interval specified in anomaly.detection.interval.ms. This is only used when kafka.broker.failure.detection.enable is set to 'true'. |
| Goal violation | goal.violations.class | *com.linkedin.kafka.cruisecontrol.detector.GoalViolations* | The Goal Violations class that extends goal violations. |

| Anomaly type | Configuration | Value | Description |
|---|---|---|---|
| | anomaly.detection.goals | For the list of available goals, see the Configuring goals section. | The goals that the anomaly detector should detect if they are violated. |
| | goal.violation.detection.interval.ms | value of anomaly.detection.interval.ms | The interval in millisecond that goal violation detector will run to detect goal violations. If this interval time is not specified, goal violation detector will run with interval specified in anomaly.detection.interval.ms. |
| Disk failure | disk.failures.class | *com.linkedin.kafka.cruisecontrol.detector.DiskFailures* | The Disk Failures class that extends disk failures anomaly. |
| | disk.failure.detection.interval.ms | value of anomaly.detection.interval.ms | The interval in millisecond that disk failure detector will run to detect disk failures. If this interval time is not specified, disk failure detector will run with interval specified in anomaly.detection.interval.ms. |
| Topic anomaly | topic.anomaly.detection.interval.ms | value of anomaly.detection.interval.ms | The interval in millisecond that topic anomaly detector will run to detect topic anomalies. If this interval time is not specified, topic anomaly detector will run with interval specified in anomaly.detection.interval.ms. |
| | topic.anomaly.finder.class | *com.linkedin.kafka.cruisecontrol.detector.NoopTopicAnomalyFinder* | A list of topic anomaly finder classes to find the current state to identify topic anomalies. |
| Slow broker | slow.broker.bytes.in.rate.detection.threshold | *1024.0* | The bytes in rate threshold in units of kilobytes per second to determine whether to include brokers in slow broker detection. |
| | slow.broker.log.flush.time.threshold.ms | *1000.0* | The log flush time threshold in units of millisecond to determine whether to detect a broker as a slow broker. |
| | slow.broker.metric.history.percentile.threshold | *90.0* | The percentile threshold used to compare the latest metric value against historical value in slow broker detection. |
| | slow.broker.metric.history.margin | *3.0* | The margin used to compare the latest metric value against historical value in slow broker detection. |
| | slow.broker.peer.metric.percentile.threshold | *50.0* | The percentile threshold used to compare last metric value against peers' latest value in slow broker detection. |
| | slow.broker.peer.metric.margin | *10.0* | The margin used to compare last metric value against peers' latest value in slow broker detection. |
| | slow.broker.demotion.score | *5* | The score threshold to trigger a demotion for slow brokers. |
| | slow.broker.decommission.score | *50* | The score threshold to trigger a removal for slow brokers. |

| Anomaly type | Configuration | Value | Description |
|---|---|---|---|
| | slow.broker.self.healing.unfixable.ratio | *0.1* | The maximum ratio of slow brokers in the cluster to trigger self-healing operation. |
| Metric anomaly | metric.anomaly.class | *com.linkedin.kafka.cruisecontrol.detector.KafkaMetricAnomaly* | The name of the class that extends metric anomaly. |
| | metric.anomaly.detection.interval.ms | value of anomaly.detection.interval.ms | The interval in millisecond that metric anomaly detector will run to detect metric anomalies. If this interval time is not specified, the metric anomaly detector will run with the interval specified in anomaly.detection.interval.ms. |
| Maintenance event | maintenance.event.reader.class | *com.linkedin.kafka.cruisecontrol.detector.NoopMaintenanceEventReader* | A maintenance event reader class to retrieve maintenance events from the user-defined store. |
| | maintenance.event.class | *com.linkedin.kafka.cruisecontrol.detector.MaintenanceEvent* | The name of the class that extends the maintenance event. |
| | maintenance.event.enable.idempotence | *true* | The flag to indicate whether maintenance event detector will drop the duplicate maintenance events detected within the configured retention period. |
| | maintenance.event.idempotence.retention.ms | *180000* | The maximum time in ms to store events retrieved from the MaintenanceEventReader. Relevant only if idempotency is enabled (see maintenance.event.enable.idempotence). |
| | maintenance.event.max.idempotence.cache.size | *25* | The maximum number of maintenance events cached by the MaintenanceEventDetector within the past maintenance.event.idempotence.retention.ms ms. Relevant only if idempotency is enabled (see maintenance.event.enable.idempotence). |
| | maintenance.event.stop.ongoing.execution | *true* | The flag to indicate whether a maintenance event will gracefully stop the ongoing execution (if any) and wait until the execution stops before starting a fix for the anomaly. |

For more information about the Anomaly detector configurations, see the Cruise Control documentation.

8. Click Save changes.
9. Click on Action > Restart next to the Cruise Control service name to restart Cruise Control.

## Enabling self-healing using REST API

1. Open a command line tool.
2. Use ssh and connect to your cluster running Cruise Control.

```
ssh root@<your_hostname>
```

You will be prompted to provide your password.

3. Enable self-healing for the required anomaly types using the following POST command:

```
POST /kafkacruisecontrol/admin?enable_self_healing_for=[anomaly_type]
```

The following parameters must be used for anomaly_type:

- GOAL_VIOLATION
- BROKER_FAILURE
- METRIC_ANOMALY
- DISK_FAILURE
- TOPIC_ANOMALY

**Note:** In case you do not want to enable self-healing for certain anomaly types, you can disable them by using the following command:

```
POST /kafkacruisecontrol/admin?disable_self_healing_
for=[anomaly_type]
```

4. Check which anomalies are currently in use, and which are detected with the following GET command:

```
GET /kafkacruisecontrol/state
```

When reviewing the state of Cruise Control, you can check the status of Anomaly Detector at the following parameters:

- selfHealingEnabled - Anomaly type for which self-healing is enabled
- selfHealingDisabled - Anomaly type for which self healing is disabled
- recentGoalViolations - Recently detected goal violations
- recentBrokerFailures - Recently detected broker failures
- recentDiskFailures - Recently detected disk failures
- recentMetricAnomalies - Recently detected metric anomalies

## Adding self-healing goals to Cruise Control in Cloudera Manager

You need to specify the actions Cruise Control should take when detecting anomaly types by providing self-healing goals in Cloudera Manager.

### Procedure

1. Go to your cluster in Cloudera Manager.
2. Select Cruise Control from the list of Services.
3. Click on Configuration tab.
4. Search for Self-Healing Goals.
5. Add the required self-healing goals to the corresponding field.
6. Click Save changes.
7. Click on Action > Restart next to the Cruise Control service name to restart Cruise Control.