

Monitoring Apache Kudu

Date published: 2020-11-04

Date modified:



Legal Notice

© Cloudera Inc. 2025. 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

Kudu metrics.....	4
Listing available metrics.....	4
Collecting metrics through HTTP.....	4
Diagnostics logging.....	5
 Monitor cluster health with ksck.....	 6
 Report Kudu crashes using breakpad.....	 7
 Enable core dump for the Kudu service.....	 8
 Use the Charts Library with the Kudu service.....	 8

Kudu metrics

Kudu daemons expose a large number of metrics. Some metrics are associated with an entire server process, whereas others are associated with a particular tablet replica.

Listing available metrics

The full set of available metrics for a Kudu server can be dumped using a special command line flag:

```
$ kudu-tserver --dump_metrics_json
$ kudu-master --dump_metrics_json
```

This will output a large JSON document. Each metric indicates its name, label, description, units, and type. Because the output is JSON-formatted, this information can easily be parsed and fed into other tooling which collects metrics from Kudu servers.

For the complete list of metrics collected by Cloudera Manager for a Kudu service, look for the Kudu metrics listed under *Cloudera Manager Metrics*.

Related Information

[Cloudera Manager Metrics](#)

Collecting metrics through HTTP

Metrics can be collected from a server process via its HTTP interface by visiting `/metrics`. The output of this page is JSON for easy parsing by monitoring services. This endpoint accepts several GET parameters in its query string:

- `/metrics?metrics=<substring1>,<substring2>,...` - Limits the returned metrics to those which contain at least one of the provided substrings. The substrings also match entity names, so this may be used to collect metrics for a specific tablet.
- `/metrics?include_schema=1` - Includes metrics schema information such as unit, description, and label in the JSON output. This information is typically omitted to save space.
- `/metrics?compact=1` - Eliminates unnecessary whitespace from the resulting JSON, which can decrease bandwidth when fetching this page from a remote host.
- `/metrics?include_raw_histograms=1` - Include the raw buckets and values for histogram metrics, enabling accurate aggregation of percentile metrics over time and across hosts.
- `/metrics?level=info` - Limits the returned metrics based on their severity level. The levels are ordered and lower levels include the levels above them. If no level is specified, debug is used to include all metrics. The valid values are:
 - debug - Metrics that are diagnostically helpful but generally not monitored during normal operation.
 - info - Generally useful metrics that operators always want to have available but may not be monitored under normal circumstances.
 - warn - Metrics which can often indicate operational oddities, which may need more investigation.

For example:

```
$ curl -s 'http://example-ts:8050/metrics?include_schema=1&metrics=connections_accepted'
```

```
[
  {
    "type": "server",
    "id": "kudu.tabletserver",
```

```

    "attributes": {},
    "metrics": [
      {
        "name": "rpc_connections_accepted",
        "label": "RPC Connections Accepted",
        "type": "counter",
        "unit": "connections",
        "description": "Number of incoming TCP connections made to
the RPC server",
        "value": 92
      }
    ]
  }
]

```

```
$ curl -s 'http://example-ts:8050/metrics?metrics=log_append_latency'
```

```

[
  {
    "type": "tablet",
    "id": "c0ebf9fef1b847e2a83c7bd35c2056b1",
    "attributes": {
      "table_name": "lineitem",
      "partition": "hash buckets: (55), range: [(<start>), (<end>))",
      "table_id": ""
    },
    "metrics": [
      {
        "name": "log_append_latency",
        "total_count": 7498,
        "min": 4,
        "mean": 69.3649,
        "percentile_75": 29,
        "percentile_95": 38,
        "percentile_99": 45,
        "percentile_99_9": 95,
        "percentile_99_99": 167,
        "max": 367244,
        "total_sum": 520098
      }
    ]
  }
]

```

Diagnostics logging

Kudu may be configured to periodically dump all of its metrics to a local log file using the `--metrics_log_interval_msflag`. Set this flag to the interval at which metrics should be written to a diagnostics log file.

The diagnostics log will be written to the same directory as the other Kudu log files, with a similar naming format, substituting diagnostics instead of a log level like INFO. After any diagnostics log file reaches 64MB uncompressed, the log will be rolled and the previous file will be gzip-compressed.

The log file generated has three space-separated fields. The first field is the word metrics. The second field is the current timestamp in microseconds since the Unix epoch. The third is the current value of all metrics on the server, using a compact JSON encoding. The encoding is the same as the metrics fetched via HTTP described above.

Monitor cluster health with ksck

The kudu CLI includes a tool called ksck that can be used for gathering information about the state of a Kudu cluster, including checking its health. ksck will identify issues such as under-replicated tablets, unreachable tablet servers, or tablets without a leader.

ksck should be run from the command line as the Kudu admin user, and requires the full list of master addresses to be specified:

```
$ sudo -u kudu kudu cluster ksck master-01.example.com,master-02.example.com,master-03.example.com
```

To see a full list of the options available with ksck, use the --help flag. If the cluster is healthy, ksck will print information about the cluster, a success message, and return a zero (success) exit status.

```
Master Summary
-----+-----+-----
      UUID                      | Address                      | Status
-----+-----+-----
a811c07b99394df799e6650e7310f282 | master-01.example.com      | HEALTHY
b579355eaaaa446e998606bcb7e87844 | master-02.example.com      | HEALTHY
cfdcc8592711485fad32ec4eea4fbfcd | master-02.example.com      | HEALTHY

Tablet Server Summary
-----+-----+-----
      UUID                      | Address                      | Status
-----+-----+-----
a598f75345834133a39c6e51163245db | tserver-01.example.com     | HEALTHY
e05ca6b6573b4e1f9a518157c0c0c637 | tserver-02.example.com     | HEALTHY
e7e53a91fe704296b3a59ad304e7444a | tserver-03.example.com     | HEALTHY

Version Summary
Version | Servers
-----+-----
1.7.1   | all 6 server(s) checked

Summary by table
Name    | RF | Status | Total Tablets | Healthy | Recovering | Under-replicated | Unavailable
-----+-----+-----+-----+-----+-----+-----
my_table | 3  | HEALTHY | 8             | 8       | 0          | 0                | 0

Total Count
-----+-----
Masters      | 3
Tablet Servers | 3
Tables       | 1
Tablets      | 8
Replicas     | 24
OK
```

If the cluster is unhealthy, for instance if a tablet server process has stopped, ksck will report the issue(s) and return a non-zero exit status, as shown in the abbreviated snippet of ksck output below:

```
Tablet Server Summary
-----+-----+-----
      UUID                      | Address                      | Status
-----+-----+-----
a598f75345834133a39c6e51163245db | tserver-01.example.com     | HEALTHY
e05ca6b6573b4e1f9a518157c0c0c637 | tserver-02.example.com     | HEALTHY
```

```
e7e53a91fe704296b3a59ad304e7444a | tserver-03.example.com | UNAVAILABLE
Error from 127.0.0.1:7150: Network error: could not get status from server:
Client connection negotiation failed: client connection to 127.0.0.1:7150:
connect: Connection refused (error 61) (UNAVAILABLE)

... (full output elided)

-----
Errors:
-----
Network error: error fetching info from tablet servers: failed to gather i
nfo for all tablet servers: 1 of 3 had errors
Corruption: table consistency check error: 1 out of 1 table(s) are not hea
lthy

FAILED
Runtime error: ksck discovered errors
```

To verify data integrity, the optional `--checksum_scan` flag can be set, which will ensure the cluster has consistent data by scanning each tablet replica and comparing results. The `--tables` or `--tablets` flags can be used to limit the scope of the checksum scan to specific tables or tablets, respectively.

For example, checking data integrity on the `my_table` table can be done with the following command:

```
$ sudo -u kudu kudu cluster ksck --checksum_scan --tables my_table master-01
.example.com,master-02.example.com,master-03.example.com
```

By default, `ksck` will attempt to use a snapshot scan of the table, so the checksum scan can be done while writes continue.

Finally, `ksck` also supports output in JSON format using the `--ksck_format` flag. JSON output contains the same information as the plain text output, but in a format that can be used by other tools. See `kudu cluster ksck --help` for more information.

Report Kudu crashes using breakpad

Kudu uses the Google breakpad library to generate a minidump whenever Kudu experiences a crash. A minidump file contains important debugging information about the process that crashed, including shared libraries loaded and their versions, a list of threads running at the time of the crash, the state of the processor registers and a copy of the stack memory for each thread, and CPU and operating system version information. These minidumps are typically only a few MB in size and are generated even if core dump generation is disabled. Currently, generating minidumps is only possible on Linux deployments.

By default, Kudu stores its minidumps in a subdirectory of the configured `glog` directory called `minidumps`. This location can be customized by setting the `--minidump_path` flag. Kudu will retain only a certain number of minidumps before deleting the older ones, in an effort to avoid filling up the disk with minidump files. The maximum number of minidumps that will be retained can be controlled by setting the `--max_minidumps` gflag.

Minidumps contain information specific to the binary that created them and are therefore not useful without access to the exact binary that crashed, or a very similar binary.

Kudu developers can access the minidump tools in their development environment because they are installed as part of the Kudu thirdparty build. They can be found in the Kudu development environment under `uninstrumented/bin`. For example, `thirdparty/installed/uninstrumented/bin/minidump-2-core`.

If minidumps are enabled, it is possible to force Kudu to create a minidump without killing the process. To do that, send a `USR1` signal to the `kudu-tserver` or `kudu-master` process. For example:

```
sudo pkill -USR1 kudu-tserver
```

Viewing the minidump stack trace with the GNU debugger

Although a minidump contains no heap information, it does contain thread and stack information. You can convert a minidump to a core file to view it with GDB.

To convert the minidump (.dmp file) to a core file:

```
minidump-2-core -o 02cb4a97-ee37-6454-73a9d9cb-590c7dde.core \  
02cb4a97-ee37-6454-73a9d9cb-590c7dde.dmp
```

To view the core file with GDB (on a parcel deployment):

```
gdb /opt/cloudera/parcels/KUDU/lib/kudu/sbin-release/kudu-master \  
-s /opt/cloudera/parcels/KUDU/lib/debug/usr/lib/kudu/sbin-release/kudu-ma  
ster.debug \  
02cb4a97-ee37-6454-73a9d9cb-590c7dde.core
```

Enable core dump for the Kudu service

If Kudu crashes, you can use Cloudera Manager to generate a core dump to get more information about the crash.

Procedure

1. Go to the Kudu service.
2. Click the Configuration tab.
3. Search for core dump.
4. Check the checkbox for the Enable Core Dump property.
5. (Optional) Unless otherwise configured, the dump file is generated in the default core dump directory, /var/log/kudu, for both the Kudu master and the tablet servers.
 - To configure a different dump directory for the Kudu master, modify the value of the Kudu Master Core Dump Directory property.
 - To configure a different dump directory for the Kudu tablet servers, modify the value of the Kudu Tablet Server Core Dump Directory property.
6. Click Save Changes.

Use the Charts Library with the Kudu service

By default, the Status tab for the Kudu service displays a dashboard containing a limited set of charts.

About this task

For details on the terminology used in these charts, and instructions on how to query for time-series data, display chart details, and edit charts, see *Charting Time-Series Data*.

The Kudu service's Charts Library tab also displays a dashboard containing a much larger set of charts, organized by categories such as process charts, host charts, CPU charts, and so on, depending on the entity (service, role, or host) that you are viewing. You can use these charts to keep track of disk space usage, the rate at which data is being inserted/modified in Kudu across all tables, or any critical cluster events. You can also use them to keep track of individual tables. For example, to find out how much space a Kudu table is using on disk:

Procedure

1. Go to the Kudu service and navigate to the Charts Library tab.

2. On the left-hand side menu, click Tables to display the list of tables currently stored in Kudu.
3. Click on a table name to view the default dashboard for that table. The Total Tablet Size On Disk Across Kudu Replicas chart displays the total size of the table on disk using a time-series chart.

Hovering with your mouse over the line on the chart opens a small pop-up window that displays information about that data point. Click the data stream within the chart to display a larger pop-up window that includes additional information for the table at the point in time where the mouse was clicked.

Related Information

[Charting Time-Series Data](#)