Cassandra configuration config by cqlsh

Asked 2 years, 9 months ago Modified 2 years, 9 months ago Viewed 232 times

Cassandra version: `3.9`, CQLSH version: `5.0.1`

Can I query Cassandra configuration (`cassandra.yaml`) using `cqlsh`?

cassandra cql cassandra-3.0 cqlsh

Share Improve this question Follow

asked Jul 24, 2020 at 6:24

ByeBye

6,534 ● 5 ● 30 ● 63

1 Answer

Sorted by: Highest score (default)

2

No, it's not possible in your version. It's possible only starting with Cassandra 4.0 that has so-called virtual tables, and there is a special table for configurations: `system_views.settings`:

```
cqlsh:test> select * from system_views.settings ;
 name                                          | value
-----------------------------------------------+-------
     transparent_data_encryption_options_enabled | false
   transparent_data_encryption_options_iv_length |    16
                                trickle_fsync | false
                  trickle_fsync_interval_in_kb | 10240
                truncate_request_timeout_in_ms | 60000
....
```

You can find more information on the virtual tables in the following blog post from TLP.

In the meantime, you can access configuration parameters via JMX.

Share Improve this answer Follow

edited Jul 24, 2020 at 7:23

answered Jul 24, 2020 at 6:47

Alex Ott
76.7k ● 8 ● 83 ● 126

Virtual tables are coming in Cassandra 4.0

08 Mar 2019

One of the exciting features coming in Cassandra 4.0 is the addition of Virtual Tables. They will expose elements like configuration settings, metrics, or running compactions through the CQL interface instead of JMX for more convenient access. This post explains what Virtual Tables are and walks through the various types that will be available in version 4.0.

Virtual Tables

The term “Virtual Tables” can be confusing, as a quick Google search may leave one under the impression that they are views that can be created through an DDL statement. In the context of Cassandra, however, Virtual Tables will be created and managed by Cassandra itself, with no possibility of creating custom ones through CQL.

They are not to be confused with Materialized Views either, which persist data from a base table into another table with a different primary key.

For Cassandra 4.0, virtual tables will be read only, trivially exposing data as CQL rows. Such data was (and will still be) accessible through JMX, which can be cumbersome to interact with and secure.

Two new keyspaces were added in Cassandra 4.0 to support Virtual Tables: `system_views` and `system_virtual_schema`.
The latter will contain schema information on the Virtual Tables, while the former will contain the actual tables.

```
cqlsh> select * from system_virtual_schema.tables;

 keyspace_name         | table_name    | comment
-----------------------+---------------+-------------------------------
          system_views |        caches |                 system caches
          system_views |       clients |  currently connected clients
          system_views |      settings |              current settings
          system_views | sstable_tasks |         current sstable tasks
          system_views |  thread_pools |
 system_virtual_schema |       columns |   virtual column definitions
 system_virtual_schema |     keyspaces | virtual keyspace definitions
 system_virtual_schema |        tables |    virtual table definitions
```

Neither of these keyspaces can be described through the `DESCRIBE KEYSPACE` command, so listing the rows in `system_virtual_schema.tables` is the only way to discover the Virtual Tables.

The tables themselves can be described as shown here:

```
cqlsh> describe table system_views.caches

CREATE TABLE system_views.caches (
    capacity_bytes bigint PRIMARY KEY,
    entry_count int,
    hit_count bigint,
    hit_ratio double,
    name text,
    recent_hit_rate_per_second bigint,
    recent_request_rate_per_second bigint,
    request_count bigint,
    size_bytes bigint
) WITH compaction = {'class': 'None'}
    AND compression = {};
```

Available Tables in 4.0

Since Apache Cassandra 4.0 was feature freezed in September 2018, we already have the definitive list of Virtual Tables that will land in that release.

caches

The `caches` virtual table displays the list of caches involved in Cassandra' s read path. It contains all the necessary information to get an overview of their settings, usage, and efficiency:

```
cqlsh> select * from system_views.caches;

 name     | capacity_bytes | entry_count | hit_count | hit_ratio | recent_hit_rate_per_second |
recent_request_rate_per_second | request_count | size_bytes
----------+----------------+-------------+-----------+-----------+----------------------------
+--------------------------------+---------------+------------
   chunks |       95420416 |          16 |       134 |  0.864516 |                          0 |
0 |           155 |    1048576
 counters |       12582912 |           0 |         0 |       NaN |                          0 |
0 |             0 |          0
     keys |       25165824 |          18 |        84 |  0.792453 |                          0 |
0 |           106 |       1632
     rows |              0 |           0 |         0 |       NaN |                          0 |
0 |             0 |          0
```

This information is currently available through the `nodetool info` command.

clients

The `clients` virtual tables will list all connected clients, with information such as the number of issued requests or what username it is using:

```
cqlsh> select * from system_views.clients;

 address   | port  | connection_stage | driver_name | driver_version | hostname  |
protocol_version | request_count | ssl_cipher_suite | ssl_enabled | ssl_protocol | username
-----------+-------+------------------+-------------+----------------+-----------+-------------
-----+---------------+------------------+-------------+--------------+-----------
 127.0.0.1 | 61164 |            ready |        null |           null | localhost |
4 |           146 |             null |       False |         null | anonymous
 127.0.0.1 | 61165 |            ready |        null |           null | localhost |
4 |           155 |             null |       False |         null | anonymous
```

settings

The `settings` virtual table will list all configuration settings that are exposeable in the `cassandra.yaml` config file:

```
cqlsh> select * from system_views.settings limit 100;

@ Row 1
-------+-----------------------------------------------------------------------------------------------
 name  | allocate_tokens_for_keyspace
 value | null

@ Row 2
-------+-----------------------------------------------------------------------------------------------
 name  | audit_logging_options_audit_logs_dir
 value | /Users/adejanovski/.ccm/trunk/node1/logs/audit/

@ Row 3
-------+-----------------------------------------------------------------------------------------------
 name  | audit_logging_options_enabled
 value | false

@ Row 4
-------+-----------------------------------------------------------------------------------------------
 name  | audit_logging_options_excluded_categories
 value |

@ Row 5
-------+-----------------------------------------------------------------------------------------------
 name  | audit_logging_options_excluded_keyspaces
 value |
```

```
...
...
...
@ Row 17
-------+-----------------------------------------------------------------------------------------
-------------------------------------------------------
 name  | back_pressure_strategy
 value | org.apache.cassandra.net.RateBasedBackPressure{high_ratio=0.9, factor=5, flow=FAST}

@ Row 18
-------+-----------------------------------------------------------------------------------------
-------------------------------------------------------
 name  | batch_size_fail_threshold_in_kb
 value | 50

@ Row 19
-------+-----------------------------------------------------------------------------------------
-------------------------------------------------------
 name  | batch_size_warn_threshold_in_kb
 value | 5

@ Row 20
-------+-----------------------------------------------------------------------------------------
-------------------------------------------------------
 name  | batchlog_replay_throttle_in_kb
 value | 1024
...
...
```

Here, I’ ve truncated the output, as there 209 settings exposed currently. There are plans to make this table writeable so that some settings can be changed at runtime as can currently be done through JMX. Such changes, of course, would need to be persisted in `cassandra.yaml` to survive a restart of the Cassandra process.

sstable_tasks

The `sstable_tasks` virtual table will expose currently running operations on SSTables like compactions, upgradesstables, or cleanup. For example:

```
cqlsh> select * from system_views.sstable_tasks ;

 keyspace_name | table_name  | task_id                                | kind       | progress |
total     | unit
---------------+-------------+----------------------------------------+------------+----------+--
---------+-------
    tlp_stress | sensor_data | f6506ec0-3064-11e9-95e2-b3ac36f635bf | compaction | 17422218 |
127732310 | bytes
```

These informations are currently available through the `nodetool compactionstats` command.

thread_pools

The `thread_pools` virtual table will display the metrics for each thread pool in Cassandra:

```
cqlsh> select * from system_views.thread_pools ;

 name                         | active_tasks | active_tasks_limit | blocked_tasks |
blocked_tasks_all_time | completed_tasks | pending_tasks
------------------------------+--------------+--------------------+---------------+------------
------------+-----------------+---------------
              AntiEntropyStage |            0 |                  1 |             0 |
0 |               0 |             0
         CacheCleanupExecutor |            0 |                  1 |             0 |
0 |               0 |             0
           CompactionExecutor |            0 |                  2 |             0 |
0 |            3121 |             0
         CounterMutationStage |            0 |                 32 |             0 |
0 |               0 |             0
                 GossipStage |            0 |                  1 |             0 |
0 |           17040 |             0
              HintsDispatcher |            0 |                  2 |             0 |
0 |               0 |             0
        InternalResponseStage |            0 |                  8 |             0 |
0 |               0 |             0
          MemtableFlushWriter |            0 |                  2 |             0 |
0 |              20 |             0
            MemtablePostFlush |            0 |                  1 |             0 |
0 |              21 |             0
        MemtableReclaimMemory |            0 |                  1 |             0 |
0 |              20 |             0
               MigrationStage |            0 |                  1 |             0 |
0 |               0 |             0
                    MiscStage |            0 |                  1 |             0 |
0 |               0 |             0
                MutationStage |            0 |                 32 |             0 |
```

```
0 |                8 |             0
    Native-Transport-Requests |           1 |                 128 |             0 |
0 |              717 |             0
       PendingRangeCalculator |           0 |                   1 |             0 |
0 |                6 |             0
 PerDiskMemtableFlushWriter_0 |           0 |                   2 |             0 |
0 |               20 |             0
              ReadRepairStage |           0 |                   8 |             0 |
0 |                0 |             0
                   ReadStage |           0 |                  32 |             0 |
0 |               22 |             0
                  Repair-Task |           0 |          2147483647 |             0 |
0 |                0 |             0
         RequestResponseStage |           0 |                   8 |             0 |
0 |               22 |             0
                      Sampler |           0 |                   1 |             0 |
0 |                0 |             0
     SecondaryIndexManagement |           0 |                   1 |             0 |
0 |                0 |             0
           ValidationExecutor |           0 |          2147483647 |             0 |
0 |                0 |             0
            ViewBuildExecutor |           0 |                   1 |             0 |
0 |                0 |             0
            ViewMutationStage |           0 |                  32 |             0 |
0 |                0 |             0
```

This information is currently available through the `nodetool tpstats` command.

Locality

Virtual Tables, regardless of the type, contain data that is specific to each node. They are not replicated, have no associated SSTables, and querying them will return the values of the coordinator (the node that the driver chooses to coordinate the request). They will also ignore the consistency level of the queries they are sent.

When interacting with Virtual Tables through `cqlsh`, results will come from the node that `cqlsh` connected to:

```
cqlsh> consistency ALL
Consistency level set to ALL.
cqlsh> select * from system_views.caches;

 name     | capacity_bytes | entry_count | hit_count | hit_ratio | recent_hit_rate_per_second |
recent_request_rate_per_second | request_count | size_bytes
----------+----------------+-------------+-----------+-----------+----------------------------
+--------------------------------+---------------+------------
   chunks |       95420416 |          16 |       134 |  0.864516 |                          0 |
0 |           155 |    1048576
 counters |       12582912 |           0 |         0 |       NaN |                          0 |
0 |             0 |          0
     keys |       25165824 |          18 |        84 |  0.792453 |                          0 |
0 |           106 |       1632
     rows |              0 |           0 |         0 |       NaN |                          0 |
0 |             0 |          0

(4 rows)

Tracing session: 06cb2100-3060-11e9-95e2-b3ac36f635bf

 activity                                                                  | timestamp
| source    | source_elapsed | client
---------------------------------------------------------------------------+--------------------
--------+-----------+----------------+-----------
                                                        Execute CQL3 query | 2019-02-14
14:54:20.048000 | 127.0.0.1 |              0 | 127.0.0.1
 Parsing select * from system_views.caches; [Native-Transport-Requests-1] | 2019-02-14
14:54:20.049000 | 127.0.0.1 |            390 | 127.0.0.1
                       Preparing statement [Native-Transport-Requests-1] | 2019-02-14
14:54:20.049000 | 127.0.0.1 |            663 | 127.0.0.1
```

```
                                                                Request complete | 2019-02-14
14:54:20.049424 | 127.0.0.1 |          1424 | 127.0.0.1
```

When interacting through the driver, there is no simple way of selecting a single node as coordinator. The load balancing policy is responsible for this and it is set on the `Cluster` object, not on a per query basis.
For the Datastax Java Driver, a new feature was introduced to support selecting a specific node to ease up Virtual Tables access through JAVA-1917 (https://datastax-oss.atlassian.net/browse/JAVA-1917). It adds (https://github.com/datastax/java-driver/commit/fe1094a469fc81c4b900cff014c9c1bd01d4f0f6#diff-38c80221e5a12bcbd6b727ae5c4fc1dfR100) a `setNode(Node node)` method to the `Statement` class in order to forcefully designate the node responsible for the query, and "voilà".
For the record, the same feature was added to the Python driver (https://datastax-oss.atlassian.net/browse/PYTHON-993).

Beyond Apache Cassandra 4.0

The data that is currently missing from Virtual Tables are global and table level metrics such as latencies and throughputs (Cassandra exposes A LOT of table specific metrics beyond those two).
Rest assured that these are being worked on from two different approaches in CASSANDRA-14670 (https://issues.apache.org/jira/browse/CASSANDRA-14670) and CASSANDRA-14572 (https://issues.apache.org/jira/browse/CASSANDRA-14572), which were not ready in time for the feature freeze.

It will probably take some time for Virtual Tables to match the amount of data available through JMX but we are confident it will catch up eventually. Convenient and secure CQL access to runtime metrics in Apache Cassandra will tremendously ease building tools like Reaper (http://cassandra-reaper.io) which currently rely on JMX.