

DATASTAX 

Introducing Cassandra tools

Apache Cassandra:
Core Concepts, Skills, Tools

Leo Schuman, Joe Chu

Oct 20, 2014

Learning Objectives

- **Introduce and use *nodetool***
- Introduce and use *cqlsh* shell commands
- Populate and test nodes using *cassandra-stress*
- Identify additional Cassandra tools
- Configure nodes and clusters using CCM

What is *nodetool*, and how do you execute a command?

- A command-line cluster management utility
 - `install/bin/nodetool`
`nodetool -h host -p jmx_port [command] [options]`
 - commands are issued to a specified *host* and *port* number
 - *jmx_port* is configured in *cassandra-env.sh*
 - default JMX port is 7199
- Many CCM commands invoke *nodetool* indirectly
- `nodetool status` displays summary info for a node's cluster

```
dstraining@DST:/home/cassandra$  
dstraining@DST:/home/cassandra$ bin/nodetool -h 127.0.0.1 -p 7199 status  
Datacenter: datacenter1  
=====
```

Status=Up/Down					
/ State=Normal/Leaving/Joining/Moving					
-- Address	Load	Tokens	Owns	Host ID	Rack
UN 127.0.0.1	5.46 MB	256	100.0%	3f15f7d8-cf53-432e-93b9-0cccef40b583	rack1

```
dstraining@DST:/home/cassandra$
```


What are some common *nodetool* commands?

- *nodetool* supports over 60 commands, including
 - **status**: display cluster state, load, host ID, and token
 - **info**: display node memory use, disk load, uptime, and similar data
 - **ring**: display node status and cluster ring state
- **nodetool info** displays data and settings for the specified node

```
dstraining@DST:/home/cassandra$  
dstraining@DST:/home/cassandra$ bin/nodetool -h 127.0.0.1 -p 7199 info  
Token           : (invoke with -T/--tokens to see all 256 tokens)  
ID              : 3f15f7d8-cf53-432e-93b9-0cccef40b583  
Gossip active   : true  
Thrift active   : true  
Native Transport active: true  
Load            : 5.46 MB  
Generation No   : 1397250119  
Uptime (seconds) : 344  
Heap Memory (MB) : 34.85 / 1014.00  
Data Center     : datacenter1  
Rack            : rack1  
Exceptions      : 0  
Key Cache       : size 3936 (bytes), capacity 52428800 (bytes), 153 hits, 184 requests  
in seconds  
Row Cache       : size 0 (bytes), capacity 0 (bytes), 0 hits, 0 requests, NaN recent h:  
dstraining@DST:/home/cassandra$
```

What is the *nodetool ring* command?

- Displays summary state and data for all nodes in the same ring as the node targeted by the command
 - nodetool ring** aids in comparing load balance and finding downed nodes
 - status and info provide similar data in greater detail

```
dstraining@DST:/home/cassandra$
dstraining@DST:/home/cassandra$ bin/nodetool -h 127.0.0.1 -p 7199 ring
Note: Ownership information does not include topology, for complete information, specify a keyspace
```

Datcenter: datacenter1

=====						
Address	Rack	Status	State	Load	Owns	Token
127.0.0.1	rack1	Up	Normal	5.46 MB	100.00%	-9122974228705839379
127.0.0.1	rack1	Up	Normal	5.46 MB	100.00%	-9095693506534277842
127.0.0.1	rack1	Up	Normal	5.46 MB	100.00%	-9080833455279920239
127.0.0.1	rack1	Up	Normal	5.46 MB	100.00%	-8809964270654377871

What other commands does *nodetool* support?

- Additional *nodetool* commands are discussed in context during this course and the *Cassandra: Operations and Performance Tuning* course

cfhistogram	cfstats	cleanup	clear snapshot	compact	compaction history
compaction stats	decommission	describe cluster	describe ring	drain	flush
disable auto compaction	disable backup	disable binary	disable gossip	disable handoff	disable thrift
enable auto compaction	enable backup	enable binary	enable gossip	enable handoff	enable thrift
get compaction threshold	get compaction throughput	get endpoints	get sstables	get stream throughput	gossip info
invalidate key cache	invalidate row cache	join	move	net stats	pause handoff
proxy histograms	range key sample	rebuild	rebuild index	refresh	reload triggers
remove node	reset local schema	resume handoff	scrub	set cache capacity	set cache keys to save
set compaction threshold	set compaction throughput	set stream throughput	set trace probability	snapshot	status binary
status thrift	stop	tpstats	upgrade sstables		

Exercise I: Launch and use *nodetool*





Learning Objectives

- Introduce and use *nodetool*
- **Introduce and use *cqlsh* shell commands**
- Populate and test nodes using *cassandra-stress*
- Identify additional Cassandra tools
- Configure nodes and clusters using CCM

What is *cqlsh*?

- An interactive, command-line CQL utility
 - interacts with its local Cassandra instance, by default
 - supports tab completion for commands
 - *install/bin/cqlsh*

cqlsh [options] [host [port]]

```
dstraining@DST:/home/cassandra$  
dstraining@DST:/home/cassandra$ bin/cqlsh   
Connected to Test Cluster at 127.0.0.1:9042.  
[cqlsh 5.0.1 | Cassandra 2.1.0 | CQL spec 3.2.0 | Native protocol v3]  
Use HELP for help.  
cqlsh> 
```

- *-k [keyspace]*: open *cqlsh* to interact with a specified keyspace
- *-f [file_name]*: execute commands in *file_name* then exit
- *-u [user] -p [password]*: authenticate with credentials
- *-h*: display online help for *cqlsh*
- many more ...

What shell commands does *cqlsh* support?

- *cqlsh* supports
 - CQL commands for schema definition and data manipulation
 - CQL shell commands to support CQL use
- CQL is a Cassandra capability, not restricted to *cqlsh*
- CQL shell commands may only be used with *cqlsh*

Command	Description
CAPTURE	Captures command output and appends it to a file
CONSISTENCY	Shows the current consistency level, or given a level, sets it
COPY	Imports and exports CSV (comma-separated values) data
DESCRIBE	Provides information about a Cassandra cluster or data objects
EXIT	Terminates <i>cqlsh</i>
EXPAND	Displays the expanded output for query results
SHOW	Shows the Cassandra version, host, or data type assumptions
SOURCE	Executes a file containing CQL statements
TRACING	Enables or disables request tracing

How does the *source* command work?

- **SOURCE**: load and execute an external CQL file

```
cqlsh> help source
```

```
SOURCE [cqlsh only]
```

Executes a file containing CQL statements. Gives the output for each statement in turn, if any, or any errors that occur along the way.

Errors do NOT abort execution of the CQL source file.

Usage:

```
SOURCE '<file>';
```

That is, the path to the file to be executed must be given inside a string literal. The path is interpreted relative to the current working directory. The tilde shorthand notation ('~/mydir') is supported for referring to \$HOME.

See also the --file option to cqlsh.

How does the *copy* command work?

- **COPY**: copy data to or from a specified table and CSV file

```
cqlsh> help copy
```

```
COPY [cqlsh only]
```

```
COPY x FROM: Imports CSV data into a Cassandra table
```

```
COPY x TO: Exports data from a Cassandra table in CSV format.
```

```
COPY <table_name> [ ( column [, ...] ) ]  
FROM ( '<filename>' | STDIN )  
[ WITH <option>='value' [AND ...] ];
```

```
COPY <table_name> [ ( column [, ...] ) ]  
TO ( '<filename>' | STDOUT )  
[ WITH <option>='value' [AND ...] ];
```

```
Available options and defaults:
```

DELIMITER=','	- character that appears between records
QUOTE='\"'	- quoting character to be used to quote fields
ESCAPE='\\'	- character to appear before the QUOTE char when quoted
HEADER=false	- whether to ignore the first line
NULL=''	- string that represents a null value
ENCODING='utf8'	- encoding for CSV output (COPY TO only)

When entering CSV data on STDIN, you can use the sequence "\." on a line by itself to end the data input.

How does the *describe* command work?

- **DESCRIBE**: display information about a specified CQL artifact

```
cqlsh> help describe
```

```
.....  
DESCRIBE [cqlsh only]
```

```
.....  
(DESC may be used as a shorthand.)
```

Outputs information about the connected Cassandra cluster, or about the data stored on it. Use in one of the following ways:

```
DESCRIBE KEYSPACES
```

```
.....  
Output the names of all keyspaces.
```

```
DESCRIBE KEYSPACE [<keyspacename>]
```

```
.....  
Output CQL commands that could be used to recreate the given  
keyspace, and the tables in it. In some cases, as the CQL interface
```

```
DESCRIBE TABLES
```

```
.....  
Output the names of all tables in the current keyspace, or in all  
keyspaces if there is no current keyspace.
```

```
DESCRIBE TABLE <tablename>
```

```
.....  
Output CQL commands that could be used to recreate the given table.
```

```
DESCRIBE CLUSTER
```

```
.....  
Output information about the connected Cassandra cluster
```

```
DESCRIBE [FULL] SCHEMA
```

```
.....  
Output CQL commands that could be used to recreate the entire (non-system) schema.
```

How does the *show* command work?

- **SHOW**: display version, host, or session information

```
cqlsh> help show
```

```
SHOW [cqlsh only]
```

Displays information about the current cqlsh session. Can be called in the following ways:

```
SHOW VERSION
```

Shows the version and build of the connected Cassandra instance, as well as the versions of the CQL spec and the Thrift protocol that the connected Cassandra instance understands.

```
SHOW HOST
```

Shows where cqlsh is currently connected.

```
SHOW SESSION <sessionid>
```

Pretty-prints the requested tracing session.

Exercise 2: Use *source*, *copy*, *describe*, and *show* commands



Learning Objectives

- Introduce and use *nodetool*
- Introduce and use *cqlsh* shell commands
- **Populate and test nodes using *cassandra-stress***
- Identify additional Cassandra tools
- Configure nodes and clusters using CCM

What is *cassandra-stress* and why is it used?

- A command-line benchmark and load-testing utility
 - creates *Keyspace1* with tables *Standard1*, *Super1*, and *Counter3*
 - `install/tools/bin/cassandra-stress`
- `cassandra-stress [command] [options]`

```
dstraining@DST:/home/cassandra/tools/bin$ ./cassandra-stress write tries=20 n=1000000 cl=one -mode native cql3 -schema keyspace="Keyspace1" -log file=~/load 1M rows.log
Created keyspaces. Sleeping 1s for propagation.
INFO 14:15:16 Using data-center name 'datacenter1' for DCAwareRoundRobinPolicy (if this is incorrect, please provide the correct datacenter name with DCAwareRoundRobinPolicy constructor)
Connected to cluster: Test Cluster
Datacenter: datacenter1; Host: localhost/127.0.0.1; Rack: rack1
INFO 14:15:16 New Cassandra host localhost/127.0.0.1:9042 added
dstraining@DST:/home/cassandra/tools/bin$ ./cassandra-stress mixed tries=20 ratio\(\write=1,read=3\) n=100000 cl=ONE -key dist=UNIFORM\(\1..1000000\) -schema keyspace="Keyspace1" -mode native cql3 -rate threads\>=16 threads\<=256 -log file=~/mixed autorate 50r50w 1M log
INFO 14:18:56 Using data-center name 'datacenter1' for DCAwareRoundRobinPolicy (if this is incorrect, please provide the correct datacenter name with DCAwareRoundRobinPolicy constructor)
Connected to cluster: Test Cluster
Datacenter: datacenter1; Host: localhost/127.0.0.1; Rack: rack1
INFO 14:18:56 New Cassandra host localhost/127.0.0.1:9042 added
Improvement over 16 threadCount: 24%
Improvement over 24 threadCount: -13%
Improvement over 36 threadCount: 9%
Improvement over 54 threadCount: -20%
```

What are the different commands in *cassandra-stress*?

- Only one command is used at a time

Command	Description
read	Multiple concurrent reads, but first must be populated by write
write	Multiple concurrent writes against the cluster
mixed	Interleave reads and writes with configurable ratio and distribution
counter_write	Multiple concurrent update of counters
counter_read	Multiple concurrent reads of counters; must first be populated
user	Interleaving of user provided queries, with configurable ratio and distribution
help	Print help for a command or option
print	Inspect the output of a distribution definition
legacy	Legacy support mode

- The commands have additional options to customize stress

How do you interpret *cassandra-stress* output?

- Lines report for the `-log interval=? period (default 1s)`
 - `total ops`: total number of operations since start of test
 - `adj row/s`: adjusted number of rows written per second this interval
 - `op/s`: number of operations per second this interval
 - `pk/s`: number of partitions written per second this interval
 - `row/s`: number of rows written per second this interval
 - `mean`: average latency for each operation this interval
 - `med`: median latency for each operation this interval
 - `.95`: 95% of the operations completed within the displayed time this interval
 - `.99`: 99% of the operations completed within the displayed time this interval
 - `.999`: 99.9% of the operations completed within the displayed time this interval
 - `.999`: maximum amount of time needed by an operation this interval

Running WRITE with 4 threads until stderr of mean < 0.02

total ops	, adj row/s	, op/s	, pk/s	, row/s	, mean	, med	, .95	, .99	, .999	, max	, time	, stderr	, c
1010	, 987	, 987	, 987	, 987	, 3.9	, 3.5	, 7.0	, 9.6	, 11.6	, 17.4	, 1.0	, 0.00000	, 0
1888	, 840	, 840	, 840	, 840	, 4.7	, 4.5	, 7.9	, 11.6	, 16.1	, 16.1	, 2.1	, 0.00000	, 0
2784	, 900	, 888	, 888	, 888	, 4.5	, 4.3	, 6.7	, 10.7	, 21.3	, 21.3	, 3.1	, 0.05680	, 0
3694	, 897	, 897	, 897	, 897	, 4.4	, 4.3	, 6.0	, 10.7	, 18.7	, 18.7	, 4.1	, 0.03827	, 0
4599	, 897	, 897	, 897	, 897	, 4.4	, 4.0	, 8.4	, 11.5	, 15.0	, 15.0	, 5.1	, 0.02895	, 0
5593	, 1000	, 988	, 988	, 988	, 4.0	, 3.8	, 5.9	, 8.7	, 15.4	, 15.4	, 6.1	, 0.02328	, 0

How do you interpret *cassandra-stress* output?

- Output continued:

- **time**: seconds elapsed since start of test
- **stderr**: standard deviation of the mean latency this interval
- **gc: #**: number of JVM garbage collection that took place this interval
- **max ms**: maximum amount of time for one JVM garbage collection this interval
- **sum ms**: total amount of time for JVM garbage collection this interval
- **sdv ms**: standard deviation of the JVM garbage collection times this interval
- **mb**: amount of memory garbage collected this interval

Running WRITE with 4 threads until stderr of mean < 0.02

total ops	, adj row/s,	op/s,	pk/s,	row/s,	mean,	time,	stderr,	gc: #,	max ms,	sum ms,	sdv ms,	mb
1010	, 987,	987,	987,	987,	3.9,	1.0,	0.00000,	0,	0,	0,	0,	0
1888	, 840,	840,	840,	840,	4.7,	2.1,	0.00000,	0,	0,	0,	0,	0
2784	, 900,	888,	888,	888,	4.5,	3.1,	0.05680,	0,	0,	0,	0,	0
3694	, 897,	897,	897,	897,	4.4,	4.1,	0.03827,	0,	0,	0,	0,	0
4599	, 897,	897,	897,	897,	4.4,	5.1,	0.02895,	0,	0,	0,	0,	0
5593	, 1000,	988,	988,	988,	4.0,	6.1,	0.02328,	0,	0,	0,	0,	0
6572	, 999,	999,	999,	999,	4.0,	7.1,	0.02478,	1,	55,	55,	0,	72
7722	, 1135,	1062,	1062,	1062,	3.7,	8.2,	0.02375,	0,	0,	0,	0,	0
9048	, 1328,	1318,	1318,	1318,	3.0,	9.2,	0.03201,	0,	0,	0,	0,	0
10442	, 1387,	1387,	1387,	1387,	2.8,	10.2,	0.04753,	0,	0,	0,	0,	0
11814	, 1372,	1364,	1364,	1364,	2.9,	11.2,	0.05441,	0,	0,	0,	0,	0
13288	, 1467,	1465,	1465,	1465,	2.7,	12.2,	0.05523,	0,	0,	0,	0,	0
14811	, 1536,	1515,	1515,	1515,	2.6,	13.2,	0.05701,	0,	0,	0,	0,	0
16322	, 1697,	1509,	1509,	1509,	2.6,	14.2,	0.05843,	1,	103,	103,	0,	74
18031	, 1691,	1691,	1691,	1691,	2.3,	15.2,	0.06190,	0,	0,	0,	0,	0

How does cassandra-stress simulate a real workload?

- Cassandra allows customizable data models and queries
 - The *user* command will allow for custom schemas, inserts and queries

```
Usage: user profile=? ops(?) [clustering=DIST(?)] [err<?] [n>?] [n<?] [no-warmup] [cl=?]
OR
Usage: user profile=? n=? ops(?) [clustering=DIST(?)] [no-warmup] [cl=?]
OR
Usage: user profile=? duration=? ops(?) [clustering=DIST(?)] [no-warmup] [cl=?]
```

profile=? Specify the path to a yaml cql3 profile

ops(null): Specify the ratios for inserts/queries to perform; e.g. ops(insert=2,<query1>=1) will perform 2 inserts for each query1
null

[clustering=DIST(?)]: Distribution clustering runs of operations of the same kind

EXP(min..max) An exponential distribution over the range [min..max]

EXTREME(min..max,shape) An extreme value (Weibull) distribution over the range [min..max]

QEXTREME(min..max,shape,quantas) An extreme value, split into quantas, within which the chance of selection is uniform

GAUSSIAN(min..max,stdvrng) A gaussian/normal distribution, where mean=(min+max)/2, and stdev is (mean-min)/stdvrng

GAUSSIAN(min..max,mean,stdev) A gaussian/normal distribution, with explicitly defined mean and stdev

UNIFORM(min..max) A uniform distribution over the range [min, max]

FIXED(val) A fixed distribution, always returning the same value

Preceding the name with ~ will invert the distribution, e.g. ~exp(1..10) will yield 10 most, instead of least, often

Aliases: extr, qextr, gauss, normal, norm, weibull

err<? (default=0.02) Run until the standard error of the mean is below this fraction

n>? (default=30) Run at least this many iterations before accepting uncertainty convergence

n<? (default=200) Run at most this many iterations before accepting uncertainty convergence

no-warmup Do not warmup the process

cl=? (default=ONE) Consistency level to use

n=? Number of operations to perform

duration=? Time to run in (in seconds, minutes or hours)

- A YAML profile can be created to customized options
 - The *profile* option is used to load the specified YAML file

Exercise 3: Write and monitor data with *cassandra-stress*



Learning Objectives

- Introduce and use *nodetool*
- Introduce and use *cqlsh* shell commands
- Populate and test nodes using *cassandra-stress*
- **Identify additional Cassandra tools**
- Configure nodes and clusters using CCM

What additional Cassandra tools are available?

- Each of these tools are distributed with Cassandra
 - `sstablekeys`: output the partition keys in a SSTable
 - `sstableloader`: bulk load external data into a cluster
 - `sstablescrub`: used with `nodetool repair` to fix corrupted tables
 - `sstable2json`, `json2sstable`: export/import tools for data inspection
 - `sstableupgrade`: upgrade SSTables to current Cassandra version
 - `sstablemetadata`: display metadata information about a SSTable
 - `sstable repairedset`: mark a SSTable as repaired
 - `sstablesplit`: split a large SSTable into smaller files
 - `token-generator`: generate tokens to manually assign to Cassandra nodes
- Some of these are introduced in context during this course, and the *Apache Cassandra: Operations and Performance Tuning* course

Learning Objectives

- Introduce and use *nodetool*
- Introduce and use *cqlsh* shell commands
- Populate and test nodes using *cassandra-stress*
- Identify additional Cassandra tools
- **Configure nodes and clusters using CCM**

What is Cassandra Cluster Manager (CCM)?

- **Creates and manages multi-node clusters on a local machine**
 - useful for configuring development and test clusters
 - communicates with *localhost* only
 - not used for configuring production clusters
- **Open source utility**
 - created by Sylvain Lebresne of DataStax
 - source code is available at <https://github.com/pcmanus/ccm>
 - requires Python 2.7+, PyYAML, Six, Ant
- **Pre-installed on the student virtual machine**
 - installation steps and references are available in the slide notes

How do you use CCM?

- CCM may target a cluster or its nodes
 - one cluster is always the current default for cluster commands

`ccm [cluster command] [options]`

 - nodes are automatically named *node1*, *node2*, *node3*, etc...

`ccm [node command] [node name] [options]`

 - Enter `ccm --help` to list commands and `ccm [command] --help` to list options
- CCM supports over 40 commands, including
 - **create**: create new cluster using specified Cassandra version
 - **list**: display list of local clusters managed by CCM
 - **populate**: add *n* nodes to current empty cluster using default options
 - **add**: add node to current cluster
 - **start**: start all nodes in current cluster
 - **status**: display up/down status for each node in specified cluster

How do you create and populate a cluster with CCM?

```
dstraining@DST: /home
dstraining@DST:/home$ ccm create cluster1 --cassandra-version 2.0.10
Downloading http://archive.apache.org/dist/cassandra/2.0.10/apache-cassandra-2.0.10-src.tar.gz
11341598 [100.00%]
Extracting /tmp/ccm-BJJmw5.tar.gz as version 2.0.10 ...
Compiling Cassandra 2.0.10 ...
Current cluster is now: cluster1
dstraining@DST:/home$ ccm create cluster2 --cassandra-version 2.1.0
Current cluster is now: cluster2
dstraining@DST:/home$ ccm list
*cluster2
  cluster1
dstraining@DST:/home$ ccm populate --nodes 3
dstraining@DST:/home$ ccm start
dstraining@DST:/home$ ccm status
node1: UP
node3: UP
node2: UP
dstraining@DST:/home$ ccm node2 stop
dstraining@DST:/home$ ccm status
node1: UP
node3: UP
node2: DOWN
dstraining@DST:/home$
```

What other commands does CCM support?

- List all CCM commands

`ccm -help`

- Display help for a specific CCM command

`ccm [command] -help`

create	add	populate	list	switch	status
remove	clear	liveset	start	stop	flush
compact	stress	updateconf	updatelog4j	cli	setdir
bulkload	setlog	scrub	show	remove	ring
drain	cleanup	repair	shuffle	sstablesplit	decommission
json	cqlsh	version	nodetool		

Exercise 4: Create multi-node cluster using CCM



Summary

- *nodetool* provides over 60 cluster-related operations
- *cqlsh* enables both CQL and CQL shell commands
- *cassandra-stress* provides benchmark and load testing tools
- *sstableloader* provides bulk data loading
- *sstablescrub* helps with table repair processes
- *sstable2json* and *json2sstable* provide data inspection
- *sstableupgrade* converts tables to the current Cassandra version
- *CCM* enables local cluster creation for development and testing

Review Questions

- What is the default JMX port and where is it configured?
- What tools and options will provide cluster status information?
- What alternatives are available for running CQL commands?
- What tools can be used to populate data in Cassandra?
- What kind of uses does CCM have?

