This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Cassandra Installation and Configuration Guide

Cassandra 1.1.x Installation/Configuration Guide

5/7/2025

Contents

Cassandra 1.1.x Installation/Configuration Guide

> **Important**
> Cassandra connectivity is recommended for Orchestration Server deployments. Orchestration Server will execute without Cassandra, but session recovery and session http redirects will not be available without connectivity to at least one configured Cassandra node.

Overview

Apache Cassandra is an open source, eventually consistent, distributed database system used by Orchestration Server to store customer session information. It is designed to be highly scalable and decentralized (such that there is no single point of failure). See Installation for step-by-step instructions for:

1. obtaining the Cassandra installation package
2. deployment of the package to a cluster of Cassandra nodes
3. configuration of the cluster
4. loading a sample schema (Orchestration)

Data Model

The Cassandra data model is divided into four basic elements. They are:

- Columns
- Column Families
- Rows
- Keyspaces

Columns

A column is the smallest element of the Cassandra data structure. It is a tuple which consists of a name, value, and timestamp.

Column Families

A column family is to Cassandra what a table is to SQL (RDBMS). Column

families store rows, and every row is referenced by a key.

Rows

A row groups related columns together. Every row is given a key and the key determines which Cassandra node the data will be stored to.

Keyspaces

A keyspace is roughly the equivalent of a schema or a database in SQL (RDBMS). Keyspaces contain collections of column families. The name of the keyspace is a first dimension of a Cassandra hash.

Example:

```
{
  "Genesys" : { // Keyspace
    "Services" : { // Column Family
      "ORS" : { // Row
        "Name" : "Orchestration", // Column
        "Version" : "8.1.2" }, // Column
      "GVP" : {
        "Name" : "Genesys Voice Platform",
        "Version" : "8.1.1" },
      "URS" : {
        "Name" : "Universal Routing Server",
        "Version" : "8.1.2" }
    }
  }
}
```

Installation

Prerequisites

Cassandra will run on both 32bit and 64bit operating systems, with 32bit or 64bit Java. The 64bit versions are preferred.

- Download the latest binary tarball from http://cassandra.apache.org/

  Note: This guide describes the installation and configuration of Cassandra version 1.1.x. Installation and Configuration of Cassandra version 2.2.5 is covered in a second guide.

- Download Commons Daemon from http://www.apache.org/dist/commons/daemon/binaries/ - this tool allows Cassandra to be run as a service/daemon.

  Note: If running Windows, make sure you download the Windows archive! The archive should contain 32bit prunsrv.exe,

the 64bit version will be found appropriate subdirectory, either amd64 or ia64. Also, currently, versions 1.0.14 and older work correctly, but version 1.0.15 fails to properly stop the service.

- Install JDK/JRE 7 - Preferably the most stable version available.

Step-by-Step Instructions

> **Note:** These instructions may vary for 64bit versions. The following provide 32bit versions of Java and prunsrv.

Step 1: Download Cassandra and Commons Daemon

- Download the latest stable release of Cassandra prior to 2.x, currently 1.1.12, from http://cassandra.apache.org/
- To be able to install Cassandra as a service/daemon, download Commons Daemon as noted above.

Copy the Cassandra archive (and Commons Daemon archive) to each node in the Cassandra cluster, to the desired installation directory. In this guide, we will use the following:

Windows: `C:\Cassandra`

Linux: `/cassandra`

Extract the contents of the Cassandra (and Commons Daemon) archive(s) on each node. On Windows, use WinZip 'Extract here...' or equivalent. On Linux, use `gunzip` or `tar -xvf` If following the example, the directories should be placed as such:

Windows: `C:\Cassandra\apache-cassandra-1.x.x`

Linux: `/cassandra/apache-cassandra-1.x.x`

The contents of the Commons Daemon archive may be extracted to:

`apache-cassandra-1.x.x\daemon`

Set the installation directory path to the environment variable CASSANDRA_HOME, for example:

```
set CASSANDRA_HOME=C:\Cassandra\apache-cassandra-1.x.x
or
export CASSANDRA_HOME=/cassandra/apache-cassandra-1.x.x
```

Set the JAVA_HOME environment variable to the Java JRE/JDK root, for example:

```
set JAVA_HOME=C:\Program Files (x86)\Java\jre7
or
export JAVA_HOME=/usr/java/jdk1.x.x_x
```

For Linux-like installs, edit JAVA_HOME in `%CASSANDRA_HOME%\bin\cassandra-in.sh`

Also extract Commons Daemon and rename the extracted folder from 'commons-daemon-1.x.x' to 'daemon'. The 'daemon' folder should be placed within the 'apache-cassandra-1.x.x' directory.

Create the following subdirectories in the installation directory:

- commitlog
- logs
- saved_caches
- data

The above are required Cassandra installation directories for the database commitlog, log files, cache, and keyspace filestore.

Step 2: Edit configuration files

The Cassandra distribution comes with a number of configuration files that should be edited (located in `%CASSANDRA_HOME%\conf` directory).

Step 2.1: Edit cassandra.yaml

The included cassandra.yaml contains default configurations for the Cassandra cluster. It is recommended to fill in the initial tokens for nodes in the cluster by editing **initial_token**. For the first node, the initial token can be left blank or set to 0. (Optionally, you may also let the other nodes automatically obtain an initial token if AutoBootstrap is enabled.)

AutoBootstrap:

> The **AutoBootstrap** option is available in Cassandra 0.5 and above. When enabled, new nodes will automatically bootstrap themselves on startup and will pick an initial token such that it will obtain half the keys of the node with the most disk space used. **Nodes specified as seeds will not AutoBootstrap!** However, even with AutoBootstrap, it is still recommended that you specify **initial_token** as the picking of an initial token will almost certainly result in an unbalanced ring.

Ensure that the following options are pointing to the correct paths (for cassandra versions 1.0.9 and later the directories will be created on startup. Prior to that version these paths may need to exist prior to startup. The older versions are not recommended. Paths specified below are examples):

```
data_file_directories:
    - C:\Cassandra\apache-cassandra-1.x.x\data
commitlog_directory: C:\Cassandra\apache-cassandra-1.x.x\commitlog
saved_caches_directory: C:\Cassandra\apache-cassandra-1.x.x\saved_caches
```

Also in cassandra.yaml, configure the **cluster_name**, **seeds**, **listen_address** and **rpc_address** as needed.

The **cluster_name** must be identical for all nodes within a Cassandra cluster. **seeds** must be provided as a comma-delimited list of IP addresses which new nodes will be able to contact for information about the Cassandra cluster. It is recommended that all nodes have the same list of **seeds** specified, and that all nodes be specified as seed nodes. **listen_address** is the IP address that other Cassandra nodes use to connect to this node, and the **rpc_address** is the listen address for

remote procedure calls (and clients, such as the cassandra-cli). They are defaulted to *localhost*.

Also verify that **storage_port** and **rpc_port** do not conflict with other configured services. The **storage_port**, which defaults to *7000*, is the port used by Cassandra nodes for inter-node communication. The **rpc_port**, which defaults to *9160*, is used for remote procedure calls (e.g. cassandra-cli) and the Thrift service. This is the port to use when building clients for the Cassandra API.

If Cassandra is being deployed in a multiple data center configuration, the **endpoint_snitch** should be modified from the default of **SimpleSnitch** to **PropertyFileSnitch**, where the snitch then employs the cassandra-topology.properties to determine the nodes in the cluster. There are other snitch types available, please refer to the cassandra.yaml for the descriptions of these types. If a multiple data center deployment is chosen, the schema will require the correct replication information to be provided in the strategy option pairs that represent the cluster. An example for manually loading with the **PropertyFileSnitch** is described below. For Orchestration loading, the pairs in the strategy option should be the same in the persistence configuration.

Step 2.2: Edit cassandra-env.sh and cassandra.bat

To configure Cassandra's JVM (Java Virtual Machine) and the JMX (Java Management Extensions) interface, edit `conf/cassandra-env.sh` for Linux, or `bin/cassandra.bat` for Windows.

The JMX port is used for management connections (such as nodetool). If necessary, edit the following line and ensure that there are no port conflicts with existing services:

For **cassandra-env.sh**:

```
# Specifies the default port over which Cassandra will be available for
# JMX connections.
JMX_PORT="7199"
```

For **cassandra.bat**, also set **SERVICE_JVM** to the desired Windows Service name, **PATH_PRUNSRV** to the `daemon` directory, and **PR_LOGPATH** to the `logs` directory.

```
-Dcom.sun.management.jmxremote.port=7199

...

:doInstallOperation
set SERVICE_JVM="cassandra_gre_01"
rem location of Prunsrv

set PATH_PRUNSRV=%CASSANDRA_HOME%\daemon\
# For x64 installations, (OS and JAVA) set PATH_PRUNSRV=%CASSANDRA_HOME%\daemon\amd64
set PR_LOGPATH=%CASSANDRA_HOME%\logs
```

Step 2.3: Edit log4j-server.properties

Logging options can be found in `conf/log4j-server.properties`. Update the path for the log file. For versions of cassandra 1.0.9 and later, the path will be created on startup if it does not exist:

```
log4j.appender.R.File=/var/log/cassandra/system.log
```

You may also find it helpful for testing to change the logging level from INFO to DEBUG:

```
log4j.rootLogger=INFO,stdout,R
```

Step 2B: Set up the Cassandra service (Windows)

Install the Cassandra service!

> To install: 'bin\cassandra.bat INSTALL'
>
> To uninstall: 'bin\cassandra.bat UNINSTALL'

Once installed, you will be able to find and start up the Cassandra service from the Windows Services GUI. The name of the service depends on the value of **SERVICE_JVM**.

Step 3: Start up Cassandra

Linux:

> Start up Cassandra by invoking `bin/cassandra -f`. It will start up in the foreground and will log to std-out. If you don't see any *error* or *fatal* log messages or Java stack traces, then chances are you've succeeded.
>
> Press "Control-C" to stop Cassandra.
>
> If you start up Cassandra without "-f" option, it will run in background, so you need to kill the process to stop.

Windows:

> To start the service from Windows, there are two options:
>
> 1. Use the Windows Services GUI
> 2. From commandline, in the `daemon` dir (as set above):
>
> - start: prunsrv.exe start <Cassandra Service Name>
> - stop: prunsrv.exe stop <Cassandra Service Name>

NOTE: There is currently a bug in prunsrv version 1.0.15.0 which prevents the service from being stopped. Use version 1.0.14.0 to prevent this, available from:

http://archive.apache.org/dist/commons/daemon/binaries/windows/

Step 4: Load the Schema

If you are running Orchestration versions prior to 8.1.3, and have a schema prepared, load the schema. For Orchestration version 8.1.3 and later, the schema may be loaded by Orchestration. Please refer to the Orchestration Deployment guide for the configuration options required.

From %CASSANDRA_HOME%, execute:

```
bin\cassandra-cli -host ''host-address'' --file ''path\to\schema-filename.txt''
```

This requires a Cassandra node to be running at the specified host-address. For Orchestration, see the sample schemas.

In the event that cassandra-cli does not behave as expected (java.lang.ClassNotFoundException), refer to the Troubleshooting section.

Step 5: Using nodetool

Once all Cassandra nodes have been started, we can check the status of the Cassandra cluster using nodetool.

On one of the Cassandra nodes, from %CASSANDRA_HOME%, run

```
/bin/nodetool -h <listen_address> -p <jmx_port> ring
```

The output of this command should be similar to the example found in the Useful Tools section. There should be as many addresses in the list as the number of Cassandra nodes configured. If there are fewer nodes than expected, make sure that all nodes have unique initial tokens.

Troubleshooting

cassandra-cli fails to run! 'java.lang.ClassNotFoundException'

In the event that cassandra-cli displays a 'java.lang.ClassNotFoundException' when run, ensure that it is being run from %CASSANDRA_HOME% as opposed to %CASSANDRA_HOME%\bin

If cassandra-cli still fails, consider replacing the shell scripts/batch files with the following:

cassandra-cli.bat (Windows)

```
@REM
@REM  Licensed to the Apache Software Foundation (ASF) under one or more
@REM  contributor license agreements.  See the NOTICE file distributed with
@REM  this work for additional information regarding copyright ownership.
@REM  The ASF licenses this file to You under the Apache License, Version 2.0
@REM  (the "License"); you may not use this file except in compliance with
@REM  the License.  You may obtain a copy of the License at
@REM
@REM      http://www.apache.org/licenses/LICENSE-2.0
@REM
@REM  Unless required by applicable law or agreed to in writing, software
@REM  distributed under the License is distributed on an "AS IS" BASIS,
@REM  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
@REM  See the License for the specific language governing permissions and
@REM  limitations under the License.


@echo off
if "%OS%" == "Windows_NT" setlocal

if NOT DEFINED CASSANDRA_HOME set CASSANDRA_HOME=%~dp0..
if NOT DEFINED JAVA_HOME goto :err

REM Ensure that any user defined CLASSPATH variables are not used on startup
set CLASSPATH=

REM For each jar in the CASSANDRA_HOME lib directory call append to build the CLASSPATH
```

```
variable.
for %%i in ("%CASSANDRA_HOME%\lib\*.jar") do call :append "%%i"
goto okClasspath

:append
set CLASSPATH=%CLASSPATH%;%1
goto :eof

:okClasspath
REM Include the build\classes\main directory so it works in development
set CASSANDRA_CLASSPATH=%CLASSPATH%;"%CASSANDRA_HOME%\build\classes\
main";"%CASSANDRA_HOME%\build\classes\thrift"
goto runCli

:runCli
echo Starting Cassandra Client
"%JAVA_HOME%\bin\java" -cp %CASSANDRA_CLASSPATH% org.apache.cassandra.cli.CliMain %*
goto finally

:err
echo The JAVA_HOME environment variable must be set to run this program!
pause

:finally

ENDLOCAL
```

cassandra-cli (UNIX)

```
#!/bin/sh

# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

if [ "x$CASSANDRA_INCLUDE" = "x" ]; then
    for include in /usr/share/cassandra/cassandra.in.sh \
                   /usr/local/share/cassandra/cassandra.in.sh \
                   /opt/cassandra/cassandra.in.sh \
                   ~/.cassandra.in.sh \
                   `dirname $0`/cassandra.in.sh; do
        if [ -r $include ]; then
            . $include
            break
        fi
    done
elif [ -r $CASSANDRA_INCLUDE ]; then
    . $CASSANDRA_INCLUDE
fi
```

```
# Use JAVA_HOME if set, otherwise look for java in PATH
if [ -x $JAVA_HOME/bin/java ]; then
    JAVA=$JAVA_HOME/bin/java
else
    JAVA=`which java`
fi

#if [ -z $CASSANDRA_CONF -o -z $CLASSPATH ]; then
#    echo "You must set the CASSANDRA_CONF and CLASSPATH vars" >&2
#    exit 1
#fi
#
## Special-case path variables.
#case "`uname`" in
#    CYGWIN*)
#        CLASSPATH=`cygpath -p -w "$CLASSPATH"`
#        CASSANDRA_CONF=`cygpath -p -w "$CASSANDRA_CONF"`
#    ;;
#esac

if [ -z $CLASSPATH ]; then
    echo "You must set the CLASSPATH var" >&2
    exit 1
fi

cassandra_classpath=''

for lib in lib/*.jar
do
    cassandra_classpath=";${lib}${cassandra_classpath}"
done
CLASSPATH="${CLASSPATH}${cassandra_classpath}"

$JAVA -ea -cp $CLASSPATH \
      -Xmx256M \
      -Dlog4j.configuration=log4j-tools.properties \
        org.apache.cassandra.cli.CliMain "$@"

# vi:ai sw=4 ts=4 tw=0 et
```

Configuring Cassandra

Cassandra can be configured prior to installation by editing the files located in %CASSANDRA_HOME%\conf. Within the conf directory, are cassandra.yaml, log4j-server.properties, log4j-tools.properties and other files which may be edited to tune Cassandra's performance, to customize the Cassandra cluster settings or even change logging settings.

There are currently two common Cassandra cluster types, single cluster or multiple data center clusters. The following describes the most common for Orchestration deployments, a single cluster, using the placement strategy class SimpleStrategy. For multiple data center clusters, the placement strategy class will be NetworkTopologyStrategy, with the additional requirement of providing the data center node placement in the cassandra-topology.properties file located in the install conf directory.

Basic Configuration

Prior to creating a Cassandra cluster, it is important to first modify a few core settings in **cassandra.yaml**:

cluster_name:

Name of the Cassandra cluster. Must be identical for all nodes in cluster.

initial_token:

Determines node token position in ring. First node should be set to 0 and additional nodes may be left unset if they are added to an existing cluster. If *AutoBootstrap* is enabled, added nodes will pick a token such that it will split half the keys from the node with the most disk space used. Note that when using automatic token picking, one can only bootstrap as many nodes as the size of the existing cluster. To explicitly specify tokens, consider using a Token Generating Tool.

data_file_directories:
commitlog_directory:
saved_caches_directory:

Ensure that the above are all pointing towards valid directories.

seeds: (default: "127.0.0.1")

Specifies a comma-delimited list of IP addresses. New nodes will contact the seed nodes to determine the ring topology and to obtain gossip information about other nodes in the cluster. Every node should have the same list of seeds. If the local node is specified in its own list of seeds, it will **not** AutoBootstrap.

listen_address: (default: localhost)

The IP address that other Cassandra nodes will use to connect to this node. If left blank, uses the hostname configuration of the node.

rpc_address: (default: localhost)

The listen address for remote procedure calls. To listen on all interfaces, set to 0.0.0.0. If left blank, uses the hostname configuration of the node.

rpc_port: (default: 9160)

The port for remote procedure calls and the Thrift service. cassandra-cli uses this port.

storage_port: (default: 7000)

> The port for inter-node communication.

endpoint_snitch: (default: SimpleSnitch)

> This option determines how Cassandra views the cluster, SimpleSnitch for a single cluster and PropertyFileSnitch, or other snitch chosen in the yaml, for a multiple data center cluster.

Note that the PropertyFileSnitch requires the cassandra-topology.properties file to describe the multiple data center cluster, for example within that file the following will need to be provided:

```
# Cassandra Node IP=Data Center:Rack
135.225.58.81=DC1:RAC1

135.225.58.82=DC1:RAC2

135.225.58.83=DC2:RAC1

135.225.58.90=DC2:RAC2
```

Note also that the schema definition for multiple data center clusters differs, if the schema is being manually loaded, the format for the keyspace definition also differs. Refer to the section on Sample Orchestration Schema for Cassandra 1.1.x.

Next, modify %CASSANDRA_HOME%\bin**cassandra.bat** (if Windows), or %CASSANDRA_HOME%/conf/**cassandra-env.sh** (if Unix-based) to configure the JVM. It is important to verify that the JMX port does not conflict with other configured services:

```
-Dcom.sun.management.jmxremote.port=7199 (in cassandra.bat)
JMX_PORT="7199" (in cassandra-env.sh)
```

In addition, it may be helpful to configure logging to aid debugging. Find and update the following line in **log4j-server.properties**:

```
log4j.appender.R.File=/var/log/cassandra/system.log
```

It may also be desirable to change the logging level from INFO to DEBUG:

```
log4j.rootLogger=INFO,stdout,R
```

Sample cassandra.yaml entries for a 4 node cluster

For a 4 node cluster with ipv4 addresses:

- 135.225.58.81
- 135.225.58.82
- 135.225.58.83
- 135.225.58.90

the cassandra.yaml entries for each node follow:

```
135.225.58.81:

      cluster_name: 'Orchestration PerfTest Cluster'
      initial_token: '0'
      data_file_directories:
            - /cassandra/apache-cassandra-0.7.9/data
      commitlog_directory: /cassandra/apache-cassandra-0.7.9/commitlog
      saved_caches_directory: /cassandra/apache-cassandra-0.7.9/saved_caches
      seeds: "135.225.58.81,135.225.58.82,135.225.58.83,135.225.58.90"

      listen_address: 135.225.58.81
      rpc_address: 135.225.58.81


135.225.58.82:


      cluster_name: 'Orchestration PerfTest Cluster'
      initial_token: '42535295865117307932921825928971026431'
      data_file_directories:
            - /cassandra/apache-cassandra-0.7.9/data
      commitlog_directory: /cassandra/apache-cassandra-0.7.9/commitlog
      saved_caches_directory: /cassandra/apache-cassandra-0.7.9/saved_caches
      seeds: "135.225.58.81,135.225.58.82,135.225.58.83,135.225.58.90"
      listen_address: 135.225.58.82
      rpc_address: 135.225.58.82



135.225.58.83:


      cluster_name: 'Orchestration PerfTest Cluster'
      initial_token: '85070591730234615865843651857942052863'
      data_file_directories:
            - /cassandra/apache-cassandra-0.7.9/data
      commitlog_directory: /cassandra/apache-cassandra-0.7.9/commitlog
      saved_caches_directory: /cassandra/apache-cassandra-0.7.9/saved_caches
      seeds: "135.225.58.81,135.225.58.82,135.225.58.83,135.225.58.90"
      listen_address: 135.225.58.83
      rpc_address: 135.225.58.83


135.225.58.90:


      cluster_name: 'Orchestration PerfTest Cluster'
      initial_token: '127605887595351923798765477786913079295'
      data_file_directories:
            - /cassandra/apache-cassandra-0.7.9/data
      commitlog_directory: /cassandra/apache-cassandra-0.7.9/commitlog
      saved_caches_directory: /cassandra/apache-cassandra-0.7.9/saved_caches
      seeds: "135.225.58.81,135.225.58.82,135.225.58.83,135.225.58.90"
      listen_address: 135.225.58.90
      rpc_address: 135.225.58.90
```

Common Configurations

Sample information for one node cluster:

```
      - initial_token '0'
      - replication_factor = 1, all data is stored on one node, no tolerance of node failure.
```

Sample information for two node cluster:

```
      - initial_tokens '0', '85070591730234615865843651857942052863'
      - replication_factor = 1, half of the data is stored on each node, if one node fails,
half of the data will be inaccessible, no tolerance of node failure.
      - replication_factor = 2, all of the data is stored on each node, if one node fails,
the cluster will continue to operate, tolerance of one node failure.
```

Sample information for three node cluster:

```
      - initial_tokens '0', '56713727820156410577229101238628035242',
'113427455640312821154458202477256070484'
      - replication_factor = 1, a third of the data is stored on each node, if one node
fails, one third of the data will be inaccessible, no tolerance of node failure.
      - replication_factor = 2, two thirds of the data is stored on each node, if one node
fails, the cluster will continue to operate, tolerance of one node failure.
      - replication_factor = 3, all of the data is stored on each node, if one or two nodes
fail, the cluster will continue to operate, tolerance of two node failure.
```

Storage Schema

Creating a schema can be done over the Cassandra CLI or CQL shell. Seen below is a schema example for Orchestration on Cassandra 1.x (note that it conforms to the cassandra-cli syntax). Note that the replication factor is set to 1 in this sample and is the only allowed value for a single node deployment.

For a multiple node cassandra cluster this should be increased to increase availability. Refer to the following web site to determine the replication factor required for your deployment, noting that Orchestration performs all operations at consistency level of ONE.

> http://www.ecyrd.com/cassandracalculator/

For more discussion on this topic, please refer to http://www.datastax.com/docs/1.1/dml/data_consistency for a discussion of consistency and the replication factor (RF).

Sample Orchestration Schema for Cassandra 1.1.x and Orchestration versions prior to 8.1.3

```
/*This file contains an example of the Orchestration keyspace, which should be tailored to
the deployed cassandra instance capabilities.  This file should be copied to the cassandra
install
conf directory.

The schema can be loaded using the cassandra-cli command line interface from the cassandra
root
install directory as follows:

        ./bin/cassandra-cli -host ip-address-of-cassandra-host --file conf/orchestration-
schema.txt

        where ip-address-of-cassandra-host is the ip form of the host - ie. 135.225.58.81
        note that the above assumes that the thrift port is the default of 9160.

The cassandra-cli includes online help that explains the statements below. You can
accessed the help without connecting to a running cassandra instance by starting the
client and typing "help;"
```

```
For use with cassandra versions 1.1.x

Note that for multiple data center clusters the keyspace definition differs, which indicates
that there will be 2 replicas in DC1 and 1 in DC2:

create keyspace Orchestration
    with strategy_options={DC1:2,DC2:1}
    and placement_strategy = 'org.apache.cassandra.locator.NetworkTopologyStrategy';
*/

create keyspace Orchestration
    with strategy_options={replication_factor:1}
    and placement_strategy = 'org.apache.cassandra.locator.SimpleStrategy';

use Orchestration;

create column family Document
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 21600
    and comment = 'JSON form of the scxml document, keyed by md5 of document';

create column family Session
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 21600
    and comment = 'JSON form of the session, keyed by session GUUID';

create column family ScheduleByTimeInterval
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 21600
    and comment = 'Column names are the concatenation of scheduled ActionGUUID, action type,
and idealtime in msecs, column values are the action content. The keys are in form of time
since the epoch in msecs divided by some time increment, say 60000, for 1 minute intervals.';

create column family ScheduleBySessionID
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 21600
    and comment = 'Column names are the concatenation of scheduled ActionGUUID, action type,
and idealtime in msecs, column values are the idealtime in msecs, keyed by session id';
```

```
create column family SessionIDServerInfo
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 21600
    and comment = 'Session id and assigned node, keyed by session id';

create column family SessionIDServerInfoRIndex
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 21600
    and comment = 'Columns are session ids and the column values are also the session id,
keyed by the string form of the server node which owns the session.';

create column family SessionDetailRecords
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 21600
    and comment = 'Columns are the session id, the column value is the json representation of
Session Detail Record, keyed by sessionID.';

create column family ORS8110000
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 21600
    and comment = 'Dummy column family to designate the Orchestration schema version.';
```

Sample Orchestration Schema for Cassandra 1.1.x and Orchestration version 8.1.3

```
/*This file contains an example of the Orchestration keyspace, which should be tailored to
the deployed cassandra instance capabilities.  This file should be copied to the cassandra
install
conf directory.

The schema can be loaded using the cassandra-cli command line interface from the cassandra
root
install directory as follows:

        ./bin/cassandra-cli -host ip-address-of-cassandra-host --file conf/orchestration-
schema.txt

        where ip-address-of-cassandra-host is the ip form of the host - ie. 135.225.58.81
```

```
        note that the above assumes that the thrift port is the default of 9160.

The cassandra-cli includes online help that explains the statements below. You can
accessed the help without connecting to a running cassandra instance by starting the
client and typing "help;"

NOTE: Please assure that the replication_factor is set correctly. Use Cassandra version
1.1.12 or later.  Versions above 1.1.x are not yet supported.
*/

create keyspace Orchestration
    with strategy_options={replication_factor:1}
    and placement_strategy = 'org.apache.cassandra.locator.SimpleStrategy';

use Orchestration;

create column family Document
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 86400
    and comment = 'JSON form of the scxml document, keyed by md5 of document';

create column family Session
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 86400
    and comment = 'JSON form of the session, keyed by session GUUID';

create column family ScheduleByTimeInterval
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 86400
    and comment = 'Column names are the concatenation of scheduled ActionGUUID, action type,
and idealtime in msecs, column values are the action content. The keys are in form of time
since the epoch in msecs divided by some time increment, say 60000, for 1 minute intervals.';

create column family ScheduleBySessionID
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 86400
    and comment = 'Column names are the concatenation of scheduled ActionGUUID, action type,
and idealtime in msecs, column values are the idealtime in msecs, keyed by session id';
```

```
create column family SessionIDServerInfo
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 86400
    and comment = 'Session id and assigned node, keyed by session id';

create column family SessionIDServerInfoRIndex
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 86400
    and comment = 'Columns are session ids and the column values are also the session id,
keyed by the string form of the server node which owns the session.';

create column family RecoverSessionIDServerInfoRIndex
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 86400
    and comment = 'Columns are session ids and the column values are also the session id,
keyed by the string form of the server node which owns the session. Entries are only those
sessions for which recovery is enabled.';

create column family ORS8130000
    with comparator = UTF8Type
    and column_type = Standard
    and memtable_throughput = 128
    and memtable_operations = 0.29
    and read_repair_chance = 1.0
    and max_compaction_threshold = 32
    and min_compaction_threshold = 4
    and gc_grace = 86400
    and comment = 'Dummy column family to designate the Orchestration schema version.';
```

In the above examples, one may notice that during the creation of keyspaces and column families, it is possible to configure various attributes. These attributes are described in the table below:

Keyspace Attributes

Option	Default	Description
name	N/A (Required)	Name for the keyspace.
placement_strategy	org.apache.casandra.locator.SimpleStrategy	Determines how replicas will be

Option	Default	Description
		distributed among nodes in a Cassandra cluster. Allowed values: • org.apache.cassandra.locator.SimpleStrategy • org.apache.cassandra.locator.NetworkTopolog A simple strategy simply distributes replicas to the next N-1 nodes in the ring for a replication_factor of N. A network topology strategy requires the Cassandra cluster to be location-aware (able to determine location of rack/datacentre). In this case, the replication_factor is set on a per-datacentre basis.
strategy_options	N/A	Specifies configuration options for the replication strategy. For SimpleStrategy, one must specify `replication_factor:`*`number_of_replicas`*. For NetworkTopologyStrategy, one must specify *`datacentre_name:number_of_replicas`*.

Column Family Attributes

Option	Default	Description
comparator	BytesType	Defines data type to use when validating or sorting column names. The comparator cannot be changed once a column family has been created.
column_type	Standard	Determines whether column family is a regular column family or a super column family. Use `Super` for super column families.
read_repair_chance	0.1	Specifies probability that read repairs should be invoked on non-quorum reads. Value must be between 0 and 1. Lower values improve read throughput but increases chances of stale values when not using a strong consistency level.
min_compaction_threshold	4	Sets the minimum number of SSTables to trigger a minor compaction when `compaction_strategy=sizeTieredCompactionStrategy`. Raising this value causes minor compactions to start less frequently and be more I/O-intensive. Setting this value to 0 disables minor compactions.

Option	Default	Description
gc_grace_seconds	864000 (10 days)	Specifies the time to wait before garbage collecting tombstones (items marked for deletion). In a single node cluster, it can be safely set to zero.
comment	N/A	A human readable comment describing the column family.
column_metadata	N/A	Defines the attributes of a column. For each column, values for `name` and `validation_class` must be specified. It is also possible to create a secondary index for a column by setting `index_type` and `index_name`.

Performance Tuning

Besides configuring keyspaces and column families, it is possible to further tweak the performance of Cassandra by editing cassandra.yaml (Node and Cluster Configuration) or by editing cassandra-env.sh (JVM Configuration).

Descriptions of tunable properties can be found in both cassandra.yaml and cassandra-env.sh. A summary of these properties can be seen in the tables below:

Performance Tuning Properties (cassandra.yaml)

Option	Default	Description
column_index_size_in_kb	64	The size at which column indexes are added to a row. Value should be kept small if only a select few columns are consistently read from each row as a higher value implies that more row data must be deserialized for each read (until index is added).
commitlog_sync	periodic	Allowed values are `periodic` or `batch`. In `periodic` mode, the value of `commitlog_sync_period_in_ms` determines how frequently the commitlog is synchronized to disk. Writes are acknowledged at every periodic sync. If set to batch, writes are not acknowledged until fsynced to disk.
commitlog_sync_period_in_ms	10000 (10 seconds)	Determines how often (in milliseconds) to sync commitlog to disk when `commitlog_sync` is set to `periodic`.
commitlog_total_space_in_mb	4096	When commitlog reaches specified size,

Option	Default	Description
		Cassandra flushes memtables to disk for oldest commitlog segments. Reduces amount of data to replay on startup.
compaction_throughput_mb_per_sec	16	Throttles compaction to the given total throughput across entire system. Value should be proportional to rate of write throughput (16 to 32 times). Setting to 0 disables compaction throttling.
concurrent_compactors	1 (per CPU core)	Max number of concurrent compaction processes allowed on a node.
concurrent_reads	32	Recommended setting is 16 * number_of_drives. This allows enough operations to queue such that the OS and drives can reorder them and minimize disk fetches.
concurrent_writes	32	Number of concurrent writes should be proportional to number of CPU cores in system. Recommended setting is (8 * number_of_cpu_cores).
in_memory_compaction_limit_in_mb	64	Size limit for rows being compacted in memory. Larger rows spill to disk and use a slower two-pass compaction process. Recommended value is 5 to 10 percent of available Java heap size.
index_interval	128	Influences granularity of SSTable indexes in memory. Smaller value indicates higher sampling of the index files, resulting in more effective indexes at the cost of memory. Recommended value is between 128 and 512 with a large column family key cache; larger value for small rows, or smaller value to increase read performance.
memtable_flush_writers	1 per data directory	Number of memtable flush writer threads. Influences flush performance and can be increased if you have a large Java heap size and many data directories.
memtable_total_space_in_mb	1/3 of heap	Total memory used for all column family memtables on a node.
multithreaded_compaction	false	When true, each compaction operation uses one thread per SSTable being merged in addition to one thread per core. Typically only useful on nodes with

Option	Default	Description
		SSD hardware.
reduce_cache_capacity_to	0.6	Sets target max cache capacity when Java heap usage reaches threshold defined by `reduce_cache_sizes_at`. Emergency measure for preventing out-of-memory errors, along with `flush_largest_memtables_at`.
reduce_cache_sizes_at	0.85	When Java heap usage exceeds this percentage (after CMS garbage collection), Cassandra reduces the cache capacity as specified by reduce_cache_capacity_to. Set to 1.0 to disable.
sliced_buffer_size_in_kb	64	Buffer size to use for reading contiguous columns. Should match size of columns typically retrieved using query operations involving a slice predicate.
stream_throughput_outbound_megabits_per_sec	400	Max outbound throughput on a node for streaming file transfers.

JVM configuration settings *Linux: conf/cassandra-env.sh Windows: bin\cassandra.bat*

Option	Default	Description
MAX_HEAP_SIZE	Half of available physical memory	Maximum heap size for the JVM. Same value is used for minimum heap size, allowing heap to be locked in memory. Should be set in conjunction with HEAP_NEWSIZE.
HEAP_NEWSIZE	100 MB per physical CPU core	Size of young generation. Larger value leads to longer GC pause times while smaller value will typically lead to more expensive GC. Set in conjunction with MAX_HEAP_SIZE.
com.sun.management.jmxremote.port	7199	Port on which Cassandra listens for JMX connections.
com.sun.management.jmxremote.ssl	false	Enable/disable SSL for JMX.
com.sun.management.jmxremote.authenticate	false	Enable/disable remote authentication for JMX.

Option	Default	Description
-Djava.rmi.server.hostname	N/A	Sets the interface hostname or IP that JMX should use to connect. Set if you have trouble connecting.

Logging

Changes to logging are made through the log4j-server.properties and log4j-tools.properties files. Within these files, it is possible to change the default logging level (`log4j.rootLogger`), the logging handlers, log message templates (`ConversionPattern`), as well as the default log file path (`log4j.appender.R.File`).

Example:

```
# output messages into a rolling log file as well as stdout
log4j.rootLogger=DEBUG,stdout,R

# stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p %d{HH:mm:ss,SSS} %m%n

# rolling log file
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.maxFileSize=20MB
log4j.appender.R.maxBackupIndex=50
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%5p [%t] %d{ISO8601} %F (line %L) %m%n
# Edit the next line to point to your logs directory
log4j.appender.R.File=C:\Cassandra\logs\system.log

# Application logging options
#log4j.logger.org.apache.cassandra=DEBUG
#log4j.logger.org.apache.cassandra.db=DEBUG
#log4j.logger.org.apache.cassandra.service.StorageProxy=DEBUG

# Adding this to avoid thrift logging disconnect errors.
log4j.logger.org.apache.thrift.server.TNonblockingServer=ERROR
```

Useful Tools

The Cassandra archive includes several helpful tools for viewing and configuring Cassandra. Most of which can be found in the %CASSANDRA_HOME%\bin directory.

cassandra-cli

Like CQL, the Cassandra CLI client utility can also be used to interact with the Cassandra cluster. Note that some attribute names may differ between CQL and the Cassandra CLI.

A cassandra-cli guide can be found at http://www.datastax.com/docs/1.1/dml/using_cli

nodetool

The nodetool utility provides a couple of helpful features. The most important of which is the ability to view the status of a cassandra cluster, for example:

```
/cassandra/apache-cassandra-0.7.9>./bin/nodetool -h 135.225.58.81 ring
          Address    Status     State          Load     Owns Token
    135.225.58.81    Up         Normal     12.15 MB   25.00% 0
    135.225.58.82    Up         Normal     11.99 MB   25.00%
42535295865117307932921825928971026431
    135.225.58.83    Up         Normal     14.41 MB   25.00%
85070591730234615865843651857942052863
    135.225.58.90    Up         Normal      4.23 MB   25.00%
127605887595351923798765477786913079295
```

Some other useful functions include:

nodetool cleanup *[keyspace] [cf_name]*

> Run on nodes that have 'lost' part of their token range to a newly added node.

nodetool decommission
removetoken status | force | *token*

> Used to remove nodes from cluster. Use `decommission` when removing live nodes as the data will be replicated from the decommissioned node. Use `removetoken` to remove a dead node, data will be streamed from remaining replicas.

nodetool move *new_token*

> Move a target node to a given Token. More efficient that decommission+bootstrap. Should run nodetool cleanup to remove unnecessary data.

nodetool repair *keyspace [cf_name] [-pr]*

> Begins an anti-entropy node repair operation. Optionally takes a list of column families.

nodetool drain

> Flushes all memtables for a node and causes the node to stop accepting write operations. Read operations will continue to work. This is typically used before upgrading to a new version of Cassandra.

nodetool flush *keyspace [cf_name]* Flushes all memtables for a keyspace to disk, allowing the commit log to be cleared. Optionally takes a list of column families.

JConsole

Available in the jdk bin directory. For example: C:\Program Files\Java\jdk1.6.0_20\bin\jconsole.exe.

After opening, select the remote process button and enter the ip:jmx port of the cassandra node, the jmx console will display heap memory usage, live threads, classes loaded, and CPU usage. Check out the MBeans tab for info regarding the cassandra exposed information and tools. Multiple nodes may be monitored with one console, just select connection, new connection and enter the cassandra ip:jmx port for the new node.

Resetting Cassandra to initial state

In some cases, due to errors in scxml scripts being exercised, there may be sessions which are started and never terminated. Depending on interaction volume, this may cause the cassandra data to exceed the planned limits of the deployment. In this case, after removing issues with the scxml scripts which cause unterminated sessions, stop all Orchestration nodes. Then stop all cassandra nodes. Remove all entries in the yaml defined paths for the data, saved_caches, and commitlog directories. Once this is complete for all cassandra nodes in cluster, restart the cassandra nodes, and reload the schema, or allow Orchestration to load the schema (Orchestration versions 8.1.3 or later).

Maintenance Routines

Routine Node Repair

On a production cluster, it is important to routinely run the `nodetool repair` command. `nodetool repair` is used to repair inconsistencies between nodes within a cluster. Stale data is updated by pulling the latest version from other nodes. Unless Cassandra applications perform no deletes at all, it is recommended to run scheduled repairs on all nodes at least once every `gc_grace_seconds`. If this procedure is not followed, deletes may be "forgotten" in the cluster following garbage collection.

Note that `nodetool repair` is a resource-intensive task and should be scheduled for low-usage hours. Avoid running `nodetool repair` on more than one node at a time.

For examples of scripts for non-Windows platforms please search for "Automating Some Cassandra Maintenance". Note that only repair is required, compact is not recommended.

```
    @echo off
    if "%OS%" == "Windows_NT" setlocal

    pushd %~dp0..
    if NOT DEFINED CASSANDRA_HOME set CASSANDRA_HOME=%CD%
    popd
    if NOT DEFINED JAVA_HOME goto :err

    REM Ensure that any user defined CLASSPATH variables are not used on startup
    set CLASSPATH="%CASSANDRA_HOME%\conf"

    REM For each jar in the CASSANDRA_HOME lib directory call append to build the CLASSPATH
variable.
    for %%i in ("%CASSANDRA_HOME%\lib\*.jar") do call :append "%%i"
```

```
    goto okClasspath

    :append
    set CLASSPATH=%CLASSPATH%;%1
    goto :eof

    :okClasspath
    REM Include the build\classes\main directory so it works in development
    set CASSANDRA_CLASSPATH=%CLASSPATH%;"%CASSANDRA_HOME%\build\classes\
main";"%CASSANDRA_HOME%\build\classes\thrift"

    REM The default cassandra JMX_PORT is 7199 - if that was modified during installation/
configuration of cassandra
    REM update the following to reflect the correct port number
    set JMX_PORT=7199

    echo Maintenance of Orchestration on %COMPUTERNAME% beginning %DATE% %TIME%
    REM for releases prior to 8.1.3 use the following, remove the REM on the next line and
add a REM on the for line below.
    REM for %%i in ( "Document" "Session" "ScheduleByTimeInterval" "ScheduleBySessionID"
"SessionIDServerInfo" "SessionIDServerInfoRIndex" "SessionDetailRecords" ) do
call :maintColumnFamily "%%~i"
    REM for releases after 8.1.2 the following should be used:
    for %%i in ( "Document" "Session" "ScheduleByTimeInterval" "ScheduleBySessionID"
"SessionIDServerInfo" "SessionIDServerInfoRIndex" "RecoverSessionIDServerInfoRIndex" ) do
call :maintColumnFamily "%%~i"
    goto maintComplete

    :maintColumnFamily
    REM perform nodetool repair
    echo %DATE% %TIME% repair  started   on  %~1
    "%JAVA_HOME%\bin\java" -cp %CASSANDRA_CLASSPATH% -Dlog4j.configuration=log4j-
tools.properties org.apache.cassandra.tools.NodeCmd -h %COMPUTERNAME% -p %JMX_PORT% repair
Orchestration %~1 -pr
    echo %DATE% %TIME% repair  completed on  %~1
    REM perform nodetool compact
    echo %DATE% %TIME% compact started   on  %~1
    "%JAVA_HOME%\bin\java" -cp %CASSANDRA_CLASSPATH% -Dlog4j.configuration=log4j-
tools.properties org.apache.cassandra.tools.NodeCmd -h %COMPUTERNAME% -p %JMX_PORT% compact
Orchestration %~1
    echo %DATE% %TIME% compact completed on  %~1
    REM perform nodetool cleanup
    echo %DATE% %TIME% cleanup started   on  %~1
    "%JAVA_HOME%\bin\java" -cp %CASSANDRA_CLASSPATH% -Dlog4j.configuration=log4j-
tools.properties org.apache.cassandra.tools.NodeCmd -h %COMPUTERNAME% -p %JMX_PORT% cleanup
Orchestration %~1
    echo %DATE% %TIME% cleanup completed on  %~1
    goto :eof

    :maintComplete
    echo Maintenance of Orchestration on %COMPUTERNAME% completed %DATE% %TIME%
    goto finally

    :err
    echo The JAVA_HOME environment variable must be set to run this program!
    pause

    :finally

    ENDLOCAL
```

If Node Repair was not run within GCGraceSeconds

If `nodetool repair` was not run within GCGraceSeconds (default is 10 days), then you run the risk of forgotten deletes. This may lead to inconsistencies in the data returned by different nodes. Running `nodetool repair` will not correct the issue entirely.

There are three recommended methods of dealing with this scenario:

1. Treat the node with inconsistent data as 'failed' and replace it.
2. To minimize forgotten deletes, increase GCGraceSeconds for all Column Families via the CLI, perform a full repair on all nodes, and then change GCGraceSeconds back again.
3. Another option which is less effective than the previous (but easier to perform) is to run `nodetool repair` on all nodes and then perform a compaction. After which, read-repair and regular `nodetool repair` should cause the cluster to converge.

Adding Nodes to a Cluster

Prior to Cassandra 0.5, new nodes must be configured with an appropriate initial token and then started by running ./cassandra with the -b option (for bootstrapping).

Cassandra 0.5 and newer has introduced the AutoBootstrap option, which allows nodes to automatically acquire a token key on startup. However, the initial token can still be explicitly defined by the user in cassandra.yaml. In fact, this is recommended practice as AutoBootstrapping will almost guarantee that the token ring will become unbalanced. The exception is if the cluster size is being doubled, in which case, existing nodes will keep their exiting token assignments and the new nodes will be assigned tokens that bisect the existing token ranges.

Another alternative is to recalculate tokens for all the nodes. Existing nodes will need to be assigned new tokens by using the `nodetool move` command while new nodes can simply be added to the cluster with the appropriate tokens. Once all nodes have been started with their new tokens, it is recommended to run a `nodetool cleanup` to remove unused keys on all nodes.

Note that both `nodetool move` and `nodetool cleanup` are resource intensive and should be scheduled for low-usage times.

Removing Nodes

Use `nodetool decommission` to remove a live node. Use `nodetool removetoken` (to any other machine) to remove a dead node. The token ranges the old node was responsible for will be assigned to the remaining nodes. If `decommission` is used, data for the reassigned ranges will be streamed from the decommissioned node. If `removetoken` was used, data for the reassigned ranges will be streamed from the remaining replicas.

As of Cassandra 1.0, It is possible to directly replace a dead node with a new one by using the property `cassandra.replace_token=<Token>`. This can be set by using the -D option when starting the Cassandra daemon. Once bootstrap is complete, it is strongly recommended to repair the node.