# Contents

## OpsCenter Installation Guide

### About OpsCenter
- Key features

### Installing OpsCenter
- Installing OpsCenter from Debian or Ubuntu packages
  - Prerequisites
  - Installing the OpsCenter package
  - Next steps
- Installing OpsCenter from CentOS, OEL, and RHEL packages
  - Prerequisites
  - Installing the OpsCenter package
  - Next steps
- Installing OpsCenter tarball distributions on Linux and Mac OSX
  - Prerequisites
  - Installing the OpsCenter binary
  - Next steps

### Installing OpsCenter agents
- Automatically deploying agents - Packaged installations
  - Prerequisites
  - Installing agents from OpsCenter
  - Next steps
- Manually deploying agents - Tarball installations
  - Prerequisites
  - Installing agents
  - Next steps
- Manually deploying agents - Packaged installations
  - Prerequisites
  - Installing agents
  - Next steps
- Adding custom variables to agents

### Configuring OpsCenter
- Configuring user access
- About access roles
- Assigning or modifying access roles
- Removing a user
- Configuring SSL
SSL requirements ................................................................. 15
Disabling SSL in Binary Tarball Installations (Linux and Mac OSX) .................. 15
Enabling SSL in Windows installations ........................................... 16
Configuring HTTPS .................................................................... 17
Configuring OpsCenter for multiple regions ........................................ 18
Configuring events and alerts .......................................................... 19
  Enabling email alerts ............................................................... 20
  Enabling alerts posted to a URL .................................................. 20
Configuring data collection and expiration ........................................... 22
  Estimating the amount of data generated ...................................... 22
Controlling data collection ............................................................. 22
  Excluding keyspaces and column families .................................. 22
  Changing performance data expiration times ................................ 23
Advanced configuration .............................................................. 23
OpsCenter configuration properties ............................................... 23
  [agents] ssh_port ................................................................. 24
  [webserver] port .................................................................. 24
  [webserver] interface ............................................................... 24
  [logging] level ..................................................................... 24
  [stat_reporter] interval ........................................................... 24
  [authentication] passwd_file .................................................... 24
Cluster configuration properties ...................................................... 24
  Cassandra connection properties ............................................... 25
    [jmx] port ....................................................................... 25
    [cassandra] seed_hosts .......................................................... 25
    [cassandra] api_port ............................................................. 25
Metrics Collection Properties .......................................................... 25
  [cassandra_metrics] ignored_keyspaces .................................... 25
  [cassandra_metrics] ignored_column_families ............................ 25
Performance Data Expiration Properties .......................................... 25
  1min_ttl = 604800 ................................................................. 25
  5min_ttl = 2419200 ............................................................... 25
  2hr_ttl = 31536000 ............................................................... 26

Starting and stopping OpsCenter and its agents .................................. 26
Starting OpsCenter ..................................................................... 26
Restarting OpsCenter ................................................................... 26
Stopping OpsCenter ..................................................................... 26
Starting the OpsCenter agent .......................................................... 27
Restarting the OpsCenter Agent ....................................................... 27
### Definitions of alert metrics

#### Commonly watched alert metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>System performance metrics</td>
<td>34</td>
</tr>
<tr>
<td>Column family performance metrics</td>
<td>35</td>
</tr>
<tr>
<td>Network Traffic</td>
<td>34</td>
</tr>
<tr>
<td>Disk Utilization</td>
<td>34</td>
</tr>
<tr>
<td>Disk Queue Size</td>
<td>34</td>
</tr>
<tr>
<td>Disk Request Size</td>
<td>34</td>
</tr>
<tr>
<td>Disk Latency</td>
<td>34</td>
</tr>
<tr>
<td>Disk Rates</td>
<td>34</td>
</tr>
<tr>
<td>Disk Throughput</td>
<td>34</td>
</tr>
<tr>
<td>Disk Usage (%)</td>
<td>34</td>
</tr>
<tr>
<td>Disk Usage (GB)</td>
<td>34</td>
</tr>
<tr>
<td>Load</td>
<td>35</td>
</tr>
<tr>
<td>Gossip Tasks Pending</td>
<td>34</td>
</tr>
<tr>
<td>Hinted Handoff Pending</td>
<td>34</td>
</tr>
<tr>
<td>Internal Responses Pending</td>
<td>35</td>
</tr>
<tr>
<td>Migrations Pending</td>
<td>35</td>
</tr>
<tr>
<td>Misc. Tasks Pending</td>
<td>35</td>
</tr>
<tr>
<td>Request Responses Pending</td>
<td>35</td>
</tr>
<tr>
<td>Streams Pending</td>
<td>35</td>
</tr>
<tr>
<td>CF: Bloom Filter False Positive Ratio</td>
<td>36</td>
</tr>
<tr>
<td>CF: Bloom Filter False Positives</td>
<td>36</td>
</tr>
<tr>
<td>CF: Bloom Filter Space Used</td>
<td>36</td>
</tr>
<tr>
<td>CF: SSTable Count</td>
<td>36</td>
</tr>
<tr>
<td>CF: Pending Reads/Writes</td>
<td>37</td>
</tr>
<tr>
<td>CF: RowCache Requests</td>
<td>36</td>
</tr>
<tr>
<td>CF: RowCache Hits</td>
<td>36</td>
</tr>
<tr>
<td>CF: Row Cache Hit Rate</td>
<td>36</td>
</tr>
<tr>
<td>CF: SSTable Size</td>
<td>36</td>
</tr>
<tr>
<td>CF: SSTable Count</td>
<td>37</td>
</tr>
<tr>
<td>CF: KeyCache Requests</td>
<td>36</td>
</tr>
<tr>
<td>CF: KeyCache Hits</td>
<td>36</td>
</tr>
<tr>
<td>CF: KeyCache Hit Rate</td>
<td>36</td>
</tr>
<tr>
<td>CF: KeyCache Hit Rate</td>
<td>36</td>
</tr>
<tr>
<td>CF: Local Writes</td>
<td>35</td>
</tr>
<tr>
<td>CF: Local Write Latency</td>
<td>35</td>
</tr>
<tr>
<td>CF: Local Reads</td>
<td>36</td>
</tr>
<tr>
<td>CF: Local Read Latency</td>
<td>36</td>
</tr>
<tr>
<td>Load</td>
<td>37</td>
</tr>
<tr>
<td>Memory</td>
<td>37</td>
</tr>
<tr>
<td>CPU</td>
<td>38</td>
</tr>
<tr>
<td>Disk Throughput</td>
<td>38</td>
</tr>
<tr>
<td>Disk Usage (GB)</td>
<td>38</td>
</tr>
<tr>
<td>Disk Usage (%)</td>
<td>38</td>
</tr>
<tr>
<td>Disk Rates</td>
<td>38</td>
</tr>
<tr>
<td>Disk Latency</td>
<td>38</td>
</tr>
<tr>
<td>Disk Request Size</td>
<td>38</td>
</tr>
<tr>
<td>Disk Queue Size</td>
<td>39</td>
</tr>
<tr>
<td>Disk Utilization</td>
<td>39</td>
</tr>
<tr>
<td>Network Traffic</td>
<td>39</td>
</tr>
</tbody>
</table>

### Definitions of alert metrics

#### Commonly watched alert metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>System performance metrics</td>
<td>34</td>
</tr>
<tr>
<td>Column family performance metrics</td>
<td>35</td>
</tr>
<tr>
<td>Network Traffic</td>
<td>34</td>
</tr>
<tr>
<td>Disk Utilization</td>
<td>34</td>
</tr>
<tr>
<td>Disk Queue Size</td>
<td>34</td>
</tr>
<tr>
<td>Disk Request Size</td>
<td>34</td>
</tr>
<tr>
<td>Disk Latency</td>
<td>34</td>
</tr>
<tr>
<td>Disk Rates</td>
<td>34</td>
</tr>
<tr>
<td>Disk Throughput</td>
<td>34</td>
</tr>
<tr>
<td>Disk Usage (%)</td>
<td>34</td>
</tr>
<tr>
<td>Disk Usage (GB)</td>
<td>34</td>
</tr>
<tr>
<td>Load</td>
<td>35</td>
</tr>
<tr>
<td>Gossip Tasks Pending</td>
<td>34</td>
</tr>
<tr>
<td>Hinted Handoff Pending</td>
<td>34</td>
</tr>
<tr>
<td>Internal Responses Pending</td>
<td>35</td>
</tr>
<tr>
<td>Migrations Pending</td>
<td>35</td>
</tr>
<tr>
<td>Misc. Tasks Pending</td>
<td>35</td>
</tr>
<tr>
<td>Request Responses Pending</td>
<td>35</td>
</tr>
<tr>
<td>Streams Pending</td>
<td>35</td>
</tr>
<tr>
<td>CF: Bloom Filter False Positive Ratio</td>
<td>36</td>
</tr>
<tr>
<td>CF: Bloom Filter False Positives</td>
<td>36</td>
</tr>
<tr>
<td>CF: Bloom Filter Space Used</td>
<td>36</td>
</tr>
<tr>
<td>CF: SSTable Count</td>
<td>36</td>
</tr>
<tr>
<td>CF: Pending Reads/Writes</td>
<td>37</td>
</tr>
<tr>
<td>CF: RowCache Requests</td>
<td>36</td>
</tr>
<tr>
<td>CF: RowCache Hits</td>
<td>36</td>
</tr>
<tr>
<td>CF: Row Cache Hit Rate</td>
<td>36</td>
</tr>
<tr>
<td>CF: SSTable Size</td>
<td>36</td>
</tr>
<tr>
<td>CF: SSTable Count</td>
<td>37</td>
</tr>
<tr>
<td>CF: KeyCache Requests</td>
<td>36</td>
</tr>
<tr>
<td>CF: KeyCache Hits</td>
<td>36</td>
</tr>
<tr>
<td>CF: KeyCache Hit Rate</td>
<td>36</td>
</tr>
<tr>
<td>CF: KeyCache Hit Rate</td>
<td>36</td>
</tr>
<tr>
<td>CF: Local Writes</td>
<td>35</td>
</tr>
<tr>
<td>CF: Local Write Latency</td>
<td>35</td>
</tr>
<tr>
<td>CF: Local Reads</td>
<td>36</td>
</tr>
<tr>
<td>CF: Local Read Latency</td>
<td>36</td>
</tr>
<tr>
<td>Load</td>
<td>37</td>
</tr>
<tr>
<td>Memory</td>
<td>37</td>
</tr>
<tr>
<td>CPU</td>
<td>38</td>
</tr>
<tr>
<td>Disk Throughput</td>
<td>38</td>
</tr>
<tr>
<td>Disk Usage (GB)</td>
<td>38</td>
</tr>
<tr>
<td>Disk Usage (%)</td>
<td>38</td>
</tr>
<tr>
<td>Disk Rates</td>
<td>38</td>
</tr>
<tr>
<td>Disk Latency</td>
<td>38</td>
</tr>
<tr>
<td>Disk Request Size</td>
<td>38</td>
</tr>
<tr>
<td>Disk Queue Size</td>
<td>39</td>
</tr>
<tr>
<td>Disk Utilization</td>
<td>39</td>
</tr>
<tr>
<td>Network Traffic</td>
<td>39</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Advanced Cassandra alert metrics</td>
<td>39</td>
</tr>
<tr>
<td>Advanced column family alert metrics</td>
<td>41</td>
</tr>
<tr>
<td>Advanced system alert metrics</td>
<td>42</td>
</tr>
<tr>
<td>Linux metrics</td>
<td>42</td>
</tr>
<tr>
<td>Windows metrics</td>
<td>43</td>
</tr>
<tr>
<td>Mac OS X metrics</td>
<td>44</td>
</tr>
<tr>
<td>Managing backups</td>
<td>45</td>
</tr>
<tr>
<td>Scheduling a backup</td>
<td>45</td>
</tr>
<tr>
<td>Data modeling in OpsCenter</td>
<td>45</td>
</tr>
<tr>
<td>Keyspace operations</td>
<td>45</td>
</tr>
<tr>
<td>Creating a keyspace</td>
<td>46</td>
</tr>
<tr>
<td>Managing keyspaces</td>
<td>46</td>
</tr>
<tr>
<td>Column family management</td>
<td>46</td>
</tr>
<tr>
<td>Creating a dynamic column family</td>
<td>47</td>
</tr>
<tr>
<td>Managing column families</td>
<td>47</td>
</tr>
<tr>
<td>Browsing the database</td>
<td>47</td>
</tr>
<tr>
<td><strong>Upgrading OpsCenter</strong></td>
<td>48</td>
</tr>
<tr>
<td>Upgrading OpsCenter to 2.1.3</td>
<td>48</td>
</tr>
<tr>
<td>Upgrading OpsCenter agents</td>
<td>49</td>
</tr>
<tr>
<td><strong>OpsCenter reference</strong></td>
<td>49</td>
</tr>
<tr>
<td>OpsCenter and OpsCenter agent ports</td>
<td>49</td>
</tr>
<tr>
<td>Debian and Ubuntu Package install locations</td>
<td>50</td>
</tr>
<tr>
<td>CentOS, OEL, and RHEL Package install locations</td>
<td>50</td>
</tr>
<tr>
<td>Binary Tarball distribution install locations</td>
<td>51</td>
</tr>
<tr>
<td><strong>OpsCenter API</strong></td>
<td>51</td>
</tr>
<tr>
<td>Managing Clusters</td>
<td>51</td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>52</td>
</tr>
<tr>
<td>Retrieving Cluster and Node Information</td>
<td>54</td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>54</td>
</tr>
<tr>
<td>Performing Cluster Operations</td>
<td>60</td>
</tr>
<tr>
<td>Node Administration Methods</td>
<td>60</td>
</tr>
<tr>
<td>Cluster Rebalancing Methods</td>
<td>63</td>
</tr>
<tr>
<td>Request Management Methods</td>
<td>64</td>
</tr>
<tr>
<td>Managing Keyspaces and Column Families</td>
<td>64</td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>65</td>
</tr>
<tr>
<td>Retrieving Metric Data</td>
<td>69</td>
</tr>
<tr>
<td>Filtering the Metric Data Output</td>
<td>69</td>
</tr>
<tr>
<td>Metrics Attribute Key Lists</td>
<td>76</td>
</tr>
</tbody>
</table>
OpsCenter Installation Guide

OpsCenter Enterprise Edition is a browser-based user interface for monitoring, administering, and configuring multiple clusters in a single, centralized management console.

About OpsCenter

OpsCenter is a browser-based user interface for monitoring, administering, and configuring multiple clusters in a single centralized management console.

Key features

The key features of OpsCenter include:

Dashboard

- A Dashboard that displays an overview of commonly watched performance metrics
- Adding your favorite graphs to the dashboard
- An Overview that condenses the dashboards of multiple clusters (not visible when monitoring a single cluster)

Configuration and administration

- Basic cluster configuration
- Administration tasks, such as adding a cluster, using simple point-and-click actions
- Multiple cluster management from a single OpsCenter instance using agents
- Rebalancing data across a cluster when new nodes are added
- Downloadable PDF cluster report

Alerts and performance metrics

- Built-in external notification capabilities
- Alert warnings of impending issues
- Automatic backup operations, including scheduling and removing of old backups

Database management and browsing

- Creating and managing keyspaces and column families
- Deleting a column family or truncating column families to remove the data but not the column family itself
- Handling thousands of column families and browsing data in the database

Other features

- Optional HTTPS support
- Online feedback form

OpsCenter Enterprise Edition is bundled with DataStax support offerings. OpsCenter Community Edition is licensed for any type of use, but does not contain all the features available in the enterprise edition.

The OpsCenter agents are installed on the Real-time (Cassandra), Analytics (Hadoop), and Search (Solr) nodes. They use Java Management Extensions (JMX) to monitor and manage each node. Cassandra exposes a number of statistics and management operations through JMX. Using JMX, OpsCenter obtains metrics from a cluster and issues various node administration commands, such as flushing SSTables or doing a repair.
Metrics are collected every minute from Cassandra, Analytics, and Search nodes and stored in a keyspace created by OpsCenter. For more information, see Configuring data collection and expiration.

Installing OpsCenter

OpsCenter Community Edition and OpsCenter Enterprise Edition packages are available from DataStax. To download the OpsCenter Enterprise Edition, you need your DataStax registration username and password. If you are not sure of your username and password, contact DataStax Support.

This section contains the following topics about installing OpsCenter:

Installing OpsCenter from Debian or Ubuntu packages

DataStax provides OpsCenter packages for Debian and Ubuntu distributions. For a complete list of supported platforms, see OpsCenter – Supported Platforms. There are different package repositories for OpsCenter Community Edition and OpsCenter Enterprise Edition.

The OpsCenter Debian and Ubuntu packaged releases runs as a service from root. The service initialization script is located in /etc/init.d. If the machine reboots, OpsCenter restarts automatically.

Note
By downloading OpsCenter from DataStax you agree to the terms of the DataStax Enterprise EULA or the DataStax Community EULA (End User License Agreements) posted on the DataStax website.

Prerequisites

Before installing OpsCenter make sure you meet the following prerequisites:

- OpsCenter hardware requirements: computer with 2 cores and 2 GB RAM.
- Your Cassandra or DataStax Enterprise cluster is up and running.
- IP address or hostname of the machine where you are installing OpsCenter.
Installing OpsCenter

- IP address or hostnames of two or three of your nodes.
- Port number used by JMX (default is 7199).
- APT Package Manager.
- Root or sudo access on the machine where you are installing OpsCenter.
- Oracle Java SE Runtime Environment (JRE) 6 - Install the JRE. Versions earlier than 1.6.0_19 should not be used. Java 7 is not recommended.
- Python 2.6+
- OpenSSL: 0.9.8. SSL is enabled by default.

To check for the software versions:

```
$ java -version
$ python -V
$ openssl version
```

Use the exact version listed in Prerequisites. For example, the default OpenSSL on Ubuntu 11.10 1.0.0. If running openssl version shows that 1.0.0 is installed, you must install OpenSSL 0.9.8:

```
$ sudo apt-get install libssl0.9.8
```

After installing OpenSSL 0.9.8, Ubuntu 11.10 will still show 1.0.0.

**Installing the OpsCenter package**

To install OpsCenter on a Debian or Ubuntu system:

1. Open the aptitude repository source list file (/etc/apt/sources.list) for editing. For example:
   
   ```
   # vi /etc/apt/sources.list
   ```

2. In this file, add the repository for the edition you want to install. OpsCenter Enterprise Edition installations require the DataStax <username> and <password> you received in your registration confirmation email.
   
   - **OpsCenter Community (free) Edition**:
     
     ```
     deb http://debian.datastax.com/community stable main
     ```
   
   - **OpsCenter Enterprise (paid) Edition**:
     
     ```
     deb http://<username>:<password>@debian.datastax.com/enterprise stable main
     ```

3. Add the DataStax repository key to your aptitude trusted keys:
   
   ```
   # curl -L http://debian.datastax.com/debian/repo_key | apt-key add -
   ```

4. Install the OpsCenter package using the APT Package Manager:
   
   - **OpsCenter Community (free) Edition**:
     
     ```
     # apt-get update
     # apt-get install opscenter-free
     ```
   
   - **OpsCenter Enterprise (paid) Edition**:
     
     ```
     # apt-get update
     # apt-get install opscenter
     ```
5. Set the [webserver] interface to the hostname or IP address of the OpsCenter machine in the 
/etc/opscenter/opscenterd.conf file. For example:

```
[webserver]
port = 8888
interface = 10.183.170.161
```

6. Start OpsCenter:

```
sudo service opscenterd start
```

7. Install the agents:

   • (Recommended) Use OpsCenter to install the agents. See *Automatically deploying agents - Packaged installations.*
   • Manually install agents. See *Manually deploying agents - Packaged installations.*

8. To start using OpsCenter, open a browser window and go to the OpsCenter URL at http://<opscenter_host>:8888 where <opscenter_host> is the IP or hostname of the OpsCenter machine.

Next steps

- Configuring OpsCenter
- Starting and stopping OpsCenter and its agents
- OpsCenter and OpsCenter agent ports
- Debian and Ubuntu Package install locations

**Installing OpsCenter from CentOS, OEL, and RHEL packages**

DataStax provides `yum` repositories for CentOS, Oracle Enterprise Linux (OEL), and RedHat Enterprise Linux (RHEL) distributions. For a complete list of supported platforms, see *OpsCenter – Supported Platforms*. There are different package repositories for the Community and Enterprise versions of OpsCenter.

The CentOS, OEL, and RHEL OpsCenter packaged releases create an `opscenter` user. OpsCenter runs as a service and runs as this user. The service initialization script is located in `/etc/init.d`. If the OpsCenter machine reboots, OpsCenter restarts automatically.

**Note**

By downloading OpsCenter from DataStax you agree to the terms of the DataStax Enterprise EULA or the DataStax Community EULA (End User License Agreements) posted on the DataStax website.

**Prerequisites**

Before installing OpsCenter make sure you have met the following prerequisites:

   • OpsCenter hardware requirements: computer with 2 cores and 2 GB RAM.
   • Your Cassandra or DataStax Enterprise cluster is up and running.
   • IP address or hostname of the machine where you are installing OpsCenter.
   • IP address or hostnames of two or three of your nodes.
   • Port number used by JMX (default is 7199).
   • YUM package management utility.
Installing OpsCenter

- Root or sudo access on the machine where you are installing OpsCenter.
- Oracle Java SE Runtime Environment (JRE) 6 - Install the JRE. Versions earlier than 1.6.0_19 should not be used. Java 7 is not recommended.
- Python 2.6+
- The correct version of OpenSSL for your operating system.

To check for the software versions:

```
$ java -version
$ python -V
$ openssl version
```

**Installing the OpsCenter package**

To install OpsCenter on CentOS, Oracle Linux, or RHEL:

1. (CentOS 5.x/RHEL 5.x only) Make sure you have EPEL (Extra Packages for Enterprise Linux) installed. EPEL contains dependent packages, such as Python 2.6+, required by OpsCenter. To install for both 32- and 64-bit systems:

   ```
   # rpm -Uvh http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
   ```

2. Open the yum repository specification /etc/yum.repos.d for editing. For example:

   ```
   # vi /etc/yum.repos.d/datastax.repo
   ```

3. In this file, add the repository for the edition you are installing. OpsCenter Enterprise Edition installations require the DataStax `<username>` and `<password>` you received in your registration confirmation email.

   - **OpsCenter Community (free) Edition:**
     ```
     [opscenter]
     name= DataStax Repository
     baseurl=http://rpm.datastax.com/community
     enabled=1
     gpgcheck=0
     ```

   - **OpsCenter Enterprise (paid) Edition:**
     ```
     [opscenter]
     name= DataStax Repository
     baseurl=http://<username>:<password>@rpm.datastax.com/enterprise
     enabled=1
     gpgcheck=0
     ```

4. Install the OpsCenter package using yum:

   - **OpsCenter Community (free) Edition:**
     ```
     # yum install opscenter-free
     ```

   - **OpsCenter Enterprise (paid) Edition:**
     ```
     # yum install opscenter
     ```
5. Set the [webserver] interface to the hostname or IP address of the OpsCenter machine in the /etc/opscenter/opscenterd.conf file. For example:

```
[webserver]
port = 8888
interface = 10.183.170.161
```

6. Start the OpsCenter:

```
sudo service opscenterd start
```

7. Install the agents:

   - (Recommended) Use OpsCenter to install the agents. See Automatic deployment of agents - Packaged installations.
   - Manually install agents. See Manually deploying agents - Packaged installations.

8. To start using OpsCenter, open a browser window and go to the OpsCenter URL at http://<opscenter_host>:8888 where <opscenter_host> is the IP or hostname of the OpsCenter machine.

**Next steps**

- Configuring OpsCenter
- Starting and stopping OpsCenter and its agents
- OpsCenter and OpsCenter agent ports
- CentOS, OEL, and RHEL Package install locations

**Installing OpsCenter tarball distributions on Linux and Mac OS X**

DataStax provides a binary tarball for installing OpsCenter Community Edition and OpsCenter Enterprise Edition. The tarball installs OpsCenter in a single directory. This type of installation does not require root permissions.

**Note**

By downloading OpsCenter from DataStax you agree to the terms of the DataStax Enterprise EULA or the DataStax Community EULA (End User License Agreements) posted on the DataStax website.

**Prerequisites**

Before installing OpsCenter make sure you meet the following prerequisites:

- OpsCenter hardware requirements: computer with 2 cores and 2 GB RAM.
- Your Cassandra or DataStax Enterprise cluster is up and running.
- IP address or hostname of the machine where you are installing OpsCenter.
- IP address or hostnames of two or three of your nodes.
- Port number used by JMX (default is 7199).
- Oracle Java SE Runtime Environment (JRE) 6 - Install the most recently released version of the JRE 6. Versions earlier than 1.6.0_19 should not be used. Java 7 is not recommended.
- Python 2.6+
- (Linux only) SYSSTAT Utilities (for collection of I/O system metrics). When agents are installed from packaged installations, these utilities are installed automatically.
Installing OpsCenter agents

- The correct version of OpenSSL for your operating system.

To check for the software versions:

```
$ java -version
$ python -V
$ openssl version
$ iostat -V
```

**Installing the OpsCenter binary**

To install on a Linux or Mac OSX system:

1. Download the tarball distribution of OpsCenter.
   
   For the Community (free) and Enterprise (paid) editions, you can either download the packages from the DataStax website or use the `wget` command. For example:

   - **OpsCenter Community (free) Edition:***
     ```
     $ curl -OL http://downloads.datastax.com/community/opscenter.tar.gz
     ```
   
   - **OpsCenter Enterprise (paid) Edition:***
     ```
     $ curl -OL http://<username>:<password>@downloads.datastax.com/enterprise/opscenter.tar.gz
     ```

2. Unpack the distribution.

   ```
   $ tar -xzvf opscenter.tar.gz
   ```

   Files for OpsCenter and a single OpsCenter agent are now in place.

3. Set the [webserver] interface (hostname or IP address of the OpsCenter machine) and any other required configuration properties for your environment in the `opscenterd.conf` file as described in **OpsCenter configuration properties**.

4. **Start OpsCenter.**

   After a few moments, the OpsCenter connects to its agent. A tarball for deploying agents to other nodes appears in the OpsCenter installation directory. See **Manually deploying agents - Tarball installations** to deploy agents to other nodes. You cannot deploy multiple agents to other nodes automatically (by clicking Fix in the OpsCenter console).

5. To start using OpsCenter, open a browser window and go to the OpsCenter URL at `http://<opscenter_host>:8888` where `<opscenter_host>` is the IP or hostname of the OpsCenter machine.

**Next steps**

- **Manually deploying agents - Tarball installations**
- **Configuring OpsCenter**
- **Starting and stopping OpsCenter and its agents**
- **OpsCenter and OpsCenter agent ports**
- **Binary Tarball distribution install locations**

OpsCenter is automatically installed in Windows installations.

**Installing OpsCenter agents**
Installing OpsCenter agents

You need to install OpsCenter agents on each node in a cluster following the steps for your platform. On some platforms, cluster-wide installation of agents is a one-click operation in the OpsCenter console. The agents collect data and send management and administrative commands to the nodes.

**Automatically deploying agents - Packaged installations**

Use this method when you have installed OpsCenter from a package and want a quick and easy installation that provides an SSH connection between the agent and OpsCenter. This method requires root or sudo access to your cluster nodes and login credentials and installs agents using the OpsCenter Dashboard.

**Prerequisites**

- Your Cassandra or DataStax Enterprise cluster is up and running.
- OpsCenter is installed and configured.
- JMX connectivity is enabled on each node in the cluster.
- You have root or sudo access to the machines where the agents will be installed.
- You either configured the SSH port or you will accept the default SSH port (22) for node-agent communications.

**Installing agents from OpsCenter**

To install agents from OpsCenter:

1. Open a browser window and go to the OpsCenter URL at http://<opscenter_host>:8888 where <opscenter_host> is the IP or hostname of the OpsCenter machine. For example:
   - http://110.123.4.5:8888

2. When you start OpsCenter for the first time, you will be prompted to connect to your cluster:

   ![Welcome to DataStax OpsCenter!](image)

   To get started, fill in your existing cluster's information below.
   
   Cluster Hosts/IPs (newline delimited)  Help
   
   JMX Port  Thrift Port
   7199 9160
   Add credentials (optional)

   Add Cluster

3. Enter the Hostnames or IP addresses of two or three nodes in the cluster and set the JMX set JMX and Thrift ports and credentials.
4. After connecting to the cluster, start installing the agents by clicking the **Fix** link located near the top of the Dashboard.

5. In the **Install Node Agent** dialog, click **Enter Credentials**.
6. In the **Node SSH Credentials** dialog, enter a **Username** that has root privileges or sudo access to all of the nodes in your cluster, plus any other required credentials, and then click **Done**.

7. In the **Install Nodes Agent** dialog, click the **Install on all nodes**.

8. If prompted, click **Accept Fingerprint** to add a node to the known hosts for OpsCenter.

It takes a few minutes for OpsCenter to complete the agent installations. After all agents are installed, a success message is displayed.

**Next steps**

- **Adding custom variables to agents**
Installing OpsCenter agents

- **OpsCenter and OpsCenter agent ports**

**Manually deploying agents - Tarball installations**

When you **install OpsCenter** using a tarball distribution, the agent for that OpsCenter is installed automatically as part of the OpsCenter installation process. You can manually install agents on other nodes running tarball installations of Cassandra or DataStax Enterprise clusters. Use the **agent.tar.gz** file to manually deploy the agents to multiple nodes in a cluster. This file is installed in the OpsCenter installation folder after you start OpsCenter.

**Prerequisites**

- Your Cassandra or DataStax Enterprise cluster is up and running.
- OpsCenter is installed and configured on a node in the cluster.
- JMX connectivity is enabled on each node in the cluster.
- SYSSTAT Utilities (needed for the collection of I/O metrics).

**Installing agents**

To install agents in a **tarball installation**:

1. Copy the agent tar file from your existing OpsCenter installation directory to your cluster node. For example, to copy to the install location:

   ```
   $ cd <install_location>
   $ scp agent.tar.gz <user>@<node_IP>:/<install_location>
   ```

2. Log in to the node, copy the **agent.tar.gz** file to the desired location, and unpack it. For example:

   ```
   $ ssh user@<node_IP>
   $ cd <install_location>
   $ tar -xzf agent.tar.gz
   ```

   The binary package creates and **agent** directory containing the installation files.

3. Configure and start the agent:

   a. If needed get the IP address of the OpsCenter host. On the node containing the OpsCenter:

      ```
      $ hostname -i
      ```

   b. On the node where you are installing the agent, go to the **agent** directory:

      ```
      $ cd agent
      $ bin/setup <opscenter_host>
      ```

      **Note**

      Generally the agent can detect the listener IP address for the node, which is the IP address displayed for node by running **nodetool ring -h localhost**. If needed, **add <node_listen_address>** to the above command.

4. Start the OpsCenter agent:

   ```
   $ bin/opscener-agent (in the background - default)
   $ bin/opscener-agent -f (in the foreground)
   ```
Installing OpsCenter agents

Next steps

- Adding custom variables to agents
- OpsCenter and OpsCenter agent ports

Manually deploying agents - Packaged installations

If you installed OpsCenter on a cluster node using a package, you can deploy agents on other supported CentOS, Debian, OEL, RHEL, or Ubuntu nodes.

Use this method to deploy the agents:

- If you do not need an SSH connection between the agents and the OpsCenter machine.
- If you want to install the agents as part of your node deployment process.

Because this method uses the agent packages, it requires root access. After installation, the agent runs as a service that starts when the machine boots up and restarts automatically.

Prerequisites

- Your Cassandra or DataStax Enterprise cluster is up and running.
- OpsCenter is installed and configured on a node in the cluster.
- JMX connectivity is enabled on each node in the cluster.
- If you do not install using the agent Debian or RPM package, make sure that your nodes also have the SYSSTAT utility installed (needed for the collection of I/O metrics).

Installing agents

To install agents in packaged installations:

1. On the OpsCenter machine, go to the `opscenter` directory:

   ```bash
   $ cd /usr/share/opscenter
   ```

2. Copy the agent software to the home directory in your cluster node. For example:

   ```bash
   $ scp agent.tar.gz <user>@<node_IP>:~/
   ```

3. Log in to the node, go to the home directory, and unpack it. For example:

   ```bash
   $ ssh <user>@<node_IP>
   $ cd ~/
   $ tar -xzf agent.tar.gz
   ```

The binary package creates an `agent` directory containing the installation files.
Installing OpsCenter agents

4. If you **have** root access, installing the agent using the package is recommended:
   a. If needed get the IP address of the OpsCenter host. On the node containing the OpsCenter:
      
      ```
      $ hostname -i
      ```
   b. On the node where you are installing the agent, go to the `agent` directory:
      
      ```
      $ cd agent
      ```
   c. Install the agent:
      
      ```
      RHEL CentOS: bin/install_agent.sh opscenter-agent.rpm <opscenter_host>
      Debian Ubuntu: bin/install_agent.sh opscenter-agent.deb <opscenter_host>
      ```

      **Note**
      Generally the agent can detect the listener IP address for the node, which is the IP address displayed for node by running `nodetool ring -h localhost`. If needed, add `<node_listen_address>` to the above command.

5. If you **do not have** root access:
   a. If needed get the IP address of the OpsCenter host. On the node containing the OpsCenter:
      
      ```
      $ hostname -i
      ```
   b. On the node where you are installing the agent, go to the `agent` directory:
      
      ```
      $ cd agent
      ```
   c. Run the setup command and install the agent:
      
      ```
      $ bin/setup <opscenter_host>
      ```

      **Note**
      Generally the agent can detect the listener IP address for the node, which is the IP address displayed for node by running `nodetool ring -h localhost`. If needed, add `<node_listen_address>` to the above command.

   d. Start the OpsCenter agent:
      
      ```
      $ bin/opscenter-agent (in the background - default)
      $ bin/opscenter-agent -f (in the foreground)
      ```

6. Open a browser window and go to the OpsCenter console URL at `http://<opscenter_host>:8888`. For example:
   
   ```
   http://110.123.4.5:8888
   ```

**Next steps**

- **Adding custom variables to agents**
- **OpsCenter and OpsCenter agent ports**

**Adding custom variables to agents**
OpsCenter agents do not pick up the environment variables of the currently logged-in user by default. For example, if Java is not in the machine's PATH, you may notice errors in the agent log on start-up. For example:

```
nohup: cannot run command 'java': No such file or directory
```

To correct, on the Cassandra nodes where the agents are installed, create the file `/etc/default/opscenter-agent` and set the environment variables for JAVA_HOME and any other custom environment variables that the agent may need. For example:

```
JAVA_HOME=/usr/bin/java
```

### Configuring OpsCenter

The following topics provide information about configuring OpsCenter:

#### Configuring user access

By default, access control is disabled. Any user that knows the OpsCenter URL can view all objects and perform all tasks. To control access, you configure authentication for OpsCenter users by performing these tasks:

- Add users.
- Assign passwords.
- Set access roles using the set_passwd.py utility.

#### About access roles

OpsCenter provides two access roles: admin and user. Users assigned the admin role can perform these tasks:

- Create, modify, or drop keyspaces
- Create, modify, or drop column families
- Execute flush, cleanup, compact, drain, repair, move, or decommission actions on a node
- Install an OpsCenter agent on Cassandra nodes

Users assigned the user role can perform all other OpsCenter tasks.

#### Assigning or modifying access roles

The first time you assign an access role to an administrator or user, OpsCenter generates a password file and enables access control. Authentication is required to access OpsCenter for viewing objects and performing tasks.

**To create or modify access roles:**

1. Run the set_passwd.py utility. For example, to create user johndoe with admin role privileges:

   ```
   $ python /usr/share/opscenter/bin/set_passwd.py johndoe admin
   Please enter a password for 'johndoe'.
   Password:
   ```

2. After configuring authentication, restart OpsCenter:

   ```
   $ service opscenterd restart
   ```

   Restarting is required only when you create the first user (because it enables access control). No restart is required for adding, modifying, or removing users.
Removing a user

To remove a user:

1. Edit the OpsCenter password file:
   - **Packaged installs**: `/etc/opscenter/.passwds`
   - **Binary installs**: `<install_dir>/passwds`

2. Delete the line of the user that you want to remove (`<username>:<password_hash>:<role>`). For example:
   `johndoe:5e8848...42d8:admin`

   Restarting is not required to remove a user. Restarting is required to delete the password file. Deleting the password file disables access control. If you delete all users, you will not be able to access OpsCenter.

Configuring SSL

OpsCenter uses Secure Socket Layer (SSL) to encrypt the communication protocol and authenticate traffic between OpsCenter agents and the main OpsCenter daemon (Linux and Mac OSX) or the DataStax OpsCenter Service (Windows). The default SSL state depends on the operating system:

- **Linux and Mac OSX**: enabled. To disable, see Disabling SSL.
- **Windows**: disabled. To enable, see Enabling SSL.

Consider disabling SSL if you are running OpsCenter and DataStax Enterprise or DataStax Community under the following conditions:

- On a secure internal network.
- In a development environment where agents and OpsCenter run on the same computer free from network threats.
- In a situation where you are not concerned about someone listening to OpsCenter traffic.
- In automatic deployments of OpsCenter to avoid re-installation of agents. (Unless you disable SSL, installing OpsCenter generates SSL files for encryption and requires re-installation of agents.)
- On a computer that does not have the required version of OpenSSL.

If you have no need for SSL, you can simply disable the SSL option to avoid installing OpenSSL.

SSL requirements

If the SSL option is enabled, OpsCenter requires a specific version of OpenSSL for each supported operating system:

<table>
<thead>
<tr>
<th>Version</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9.8</td>
<td>CentOS 5.x, Debian, Mac OSX, Oracle Linux 5.5, RHEL 5.x, SuSe Enterprise 11.x, Ubuntu, and Windows</td>
</tr>
<tr>
<td>1.0.0</td>
<td>CentOS 6.x, Oracle Linux 6.1, and RHEL 6.x</td>
</tr>
</tbody>
</table>

To determine which version of OpenSSL is installed on a Linux or Mac OSX system, use the following command:

```
openssl version
```

Disabling SSL in Binary Tarball Installations (Linux and Mac OSX)

By default, SSL is enabled on Linux and Mac OSX installations. You modify the configuration files for OpsCenter and its agents to disable SSL on Linux and Mac OSX.

On the OpsCenter machine:
1. Go to the directory containing the OpsCenter configuration file (opscenterd.conf):
   - cd /etc/opscenter (package install)
   - cd /<install_location>/conf (binary tarball install)

2. Open opscenterd.conf, for editing. For example:
   ```
sudo vi opscenterd.conf
   ```

3. Add the following to opscenterd.conf:
   ```
[agents]
use_ssl = false
   ```

4. **Restart OpsCenter.**

   **On the agent machine:**

   1. Go to the directory containing the OpsCenter agent configuration file (address.yaml):
      ```
cd /<install_location>/conf
   ```

   2. Open address.yaml for editing. For example:
      ```
sudo vi address.yaml
   ```

   3. Add the following command and set its value to 0.
      ```
use_ssl: 0
   ```

4. **Restart the OpsCenter agent.**

**Enabling SSL in Windows installations**

By default, SSL is disabled on Windows installations. To enable SSL, you run `setup.py` (which generates the required SSL keys and certificates), modify the configuration files for OpsCenter and its agent, and then restart the DataStax OpsCenter Agent Service.

**To enable SSL:**

1. Go to the `opscenter\bin` directory:
   ```
   Program Files (x86) > DataStax Community > opscenter > bin
   ```

2. Click or double-click `setup.py` to run it.
   The `agentKeyStore` key pairs are generated and appear in `opscenter\ssl` directory.

3. Go to the `opscenter\conf` directory:
   ```
   DataStax Community > opscenter > conf
   ```

4. Open the configuration file for OpsCenter, `opscenterd.conf`, in a text editor such as Notepad.
5. In the agents section, change use_ssl from 0 to 1 (or true), and then save the file.

6. Go to the opscenter\agent\conf directory:
   **DataStack Community > opscenter > agent > conf**

7. Open the configuration file for OpsCenter agent, address.yaml, in a text editor.

8. In the address.yaml file, change the value for use_ssl from 0 to 1, and then save the file.

   ```yaml
   use_ssl: 1
   ```

9. From the Control Panel, restart the DataStax OpsCenter Agent Service.

---

**Configuring HTTPS**

You can enable or disable Hypertext Transfer Protocol Secure (HTTPS) support in OpsCenter.

**To enable HTTPS:**
1. Open the OpsCenter configuration file, opscenterd.conf, located in one of these directories:
   - Package installations: /etc/opscenter/opscenterd.conf
   - Binary tarball installations (Linux and Mac OSX): <install_location>/conf/opscenterd.conf
   - Windows installations: Program Files (x86)\DataStax Community\opscenter\conf\opscenterd.conf

2. Scroll to the [webserver] section.
   This snippet from opscenterd.conf shows the [webserver] section that you change:
   ```
   # opscenterd.conf
   [webserver]
   port = 8888
   interface = 127.0.0.1
   # The following settings can be used to enable ssl support for the opscenter
   # web application. Change these values to point to the ssl certificate and key
   # that you wish to use for your OpsCenter install, as well as the port you would like
   # to serve ssl traffic from.
   #ssl_keyfile = /var/lib/opscenter/ssl/opscenter.key
   #ssl_certfile = /var/lib/opscenter/ssl/opscenter.pem
   #ssl_port = 8443
   ```

3. Remove the comment markers (#) in front of ssl_keyfile, ssl_certfile, and ssl_port.
   You can use the default values for the ssl_keyfile and ssl_certfile or replace them with the path to your own private
   and public certificates.

4. Save opscenterd.conf and restart OpsCenter.

**Configuring OpsCenter for multiple regions**

OpsCenter 2.1.2 and later operate in multiple regions or IP forwarding deployments.

Use the following approach for deployments where a public IP forwards to a private IP on the agent, but that machine is
not aware of (that is, can't bind to) the public IP.

**To configure OpsCenter agents for multiple regions or IP forwarding:**

1. Open the address.yaml file for editing.
   The location of this file depends on the type of installation:
   - Package installations: /var/lib/opscenter-agent/conf directory
   - Binary tarball installations (Linux and Mac OSX): <install_location>/conf directory

2. Add the following option to the address.yaml:
   ```
   local_interface: (Optional) The IP that the agent uses to connect to the local Cassandra process. This is typically
   the same as listen_address, and can usually be auto-detected.
   agent_rpc_interface: The IP that the agent HTTP server listens on. In a multiple region deployment, this is
   typically a private IP.
   agent_rpc_broadcast_address: The IP that the central OpsCenter process uses to connect to the agent.
   ```
   For example:
   ```
   local_interface: 10.114.187.46
   agent_rpc_interface: 23.22.137.46
   agent_rpc_broadcast_address: 23.22.204.59
   ```

3. Repeat the above steps for each node.
### Configuring events and alerts

The OpsCenter **Event Log** page displays a continuously updated list of events and alerts. The following list reflects the most detailed logging level available for Cassandra, DataStax Enterprise, and OpsCenter events. The available levels are DEBUG(0), INFO (1), WARN (2), ERROR (3), CRITICAL (4), ALERT (5).

Data for these events is stored in the `events` and `events_timeline` column families in the OpsCenter keyspace:

<table>
<thead>
<tr>
<th>Event</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPACTATION</td>
<td>0</td>
<td>Major compaction has occurred.</td>
</tr>
<tr>
<td>CLEANUP</td>
<td>1</td>
<td>Unused keys have been removed or cleaned up.</td>
</tr>
<tr>
<td>REPAIR</td>
<td>2</td>
<td>A repair operation has been initiated.</td>
</tr>
<tr>
<td>FLUSH</td>
<td>3</td>
<td>Memtables have been flushed to disk.</td>
</tr>
<tr>
<td>DRAIN</td>
<td>4</td>
<td>The commit log has been emptied, or drained.</td>
</tr>
<tr>
<td>DECOMMISSION</td>
<td>5</td>
<td>A leaving node has streamed its data to another node.</td>
</tr>
<tr>
<td>MOVE</td>
<td>6</td>
<td>Like NODE_MOVE; a new token range has been assigned.</td>
</tr>
<tr>
<td>NODE_DOWN</td>
<td>13</td>
<td>A node has stopped responding.</td>
</tr>
<tr>
<td>NODE_UP</td>
<td>14</td>
<td>An unresponsive node has recovered.</td>
</tr>
<tr>
<td>NODE_LEFT</td>
<td>15</td>
<td>A node has left, or been removed from, the ring.</td>
</tr>
<tr>
<td>NODE_JOIN</td>
<td>16</td>
<td>A node has joined the ring.</td>
</tr>
<tr>
<td>NODE_MOVE</td>
<td>17</td>
<td>A node has been assigned a new token range (the token has moved).</td>
</tr>
<tr>
<td>OPSC_UP</td>
<td>18</td>
<td>OpsCenter has been started and is operating.</td>
</tr>
<tr>
<td>OPSC_DOWN</td>
<td>19</td>
<td>OpsCenter was stopped or stopped running.</td>
</tr>
<tr>
<td>GC</td>
<td>20</td>
<td>Java garbage collection has been initiated.</td>
</tr>
</tbody>
</table>

Optionally, you can configure OpsCenter to send alerts for selected levels of events. These alerts can be provided remotely by email, or through HTTP to a selected URL. Alerts are disabled by default.

**Note**

Alerts are triggered only by events from the OpsCenter API/UI. For example, a `nodetool move` operation submitted from the command line does not trigger an alert. However, a move operation launched using Dashboard > List View > Actions > Move controls in the OpsCenter does trigger an alert.

All alerts contain the following information about each event captured:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>api_source_ip</td>
<td>IP that originally sent the request.</td>
<td>67.169.50.240</td>
</tr>
<tr>
<td>target_node</td>
<td>Destination of a STREAMING action.</td>
<td>50.1.1.11</td>
</tr>
<tr>
<td>event_source</td>
<td>Component that caused the event.</td>
<td>OpsCenter (i.e., restart, start)</td>
</tr>
<tr>
<td>user</td>
<td>OpsCenter user that caused the event.</td>
<td>opscenter_user</td>
</tr>
<tr>
<td>time</td>
<td>Normal timestamp for the event.</td>
<td>1311025650414527</td>
</tr>
<tr>
<td>action</td>
<td>Type of event (see above table)</td>
<td>20</td>
</tr>
<tr>
<td>message</td>
<td>Description of the event.</td>
<td>Garbage Collecting node 50.1.1.13</td>
</tr>
<tr>
<td>level</td>
<td>Numerical code for the log level.</td>
<td>1</td>
</tr>
<tr>
<td>source_node</td>
<td>Node where the event originated.</td>
<td>50.1.1.13</td>
</tr>
</tbody>
</table>
Enabling email alerts

OpsCenter can post alerts to selected email addresses. To enable email alerts, you must edit the `email.conf` file and provide valid SMTP server host and port information. This file is located in the following directories:

- **Package installations:** `/etc/opscenter/event-plugins`
- **Binary tarball installations** (Linux and Mac OSX):
  `<install_location>/opscenter/conf/event-plugins`
- **Windows installations:**
  `Program Files (x86)\DataStax Community\opscenter\conf\event-plugins`

To enable email alerts:

1. Make sure that you have valid SMTP mail accounts to send and receive alerts.
2. On the OpsCenter daemon host, open the `email.conf` file for editing.
3. Set `enabled` to 1.
4. Provide valid values for your SMTP host, port, user, and password.
5. Enable Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocol on your system if you want secure communications. Typically, SSL is required.
6. Provide valid values for the `to_addr` and `from_addr` email addresses. The `to_addr` value is the account that will receive alerts.
7. Optionally, set the level of alerts to send and the desired subject line.
8. Save `email.conf` and restart the OpsCenter daemon.

For example, in a system with email alerts enabled for critical and alert-level events, `email.conf` looks like:

```plaintext
[email]
enabled=1
# levels can be comma delimited list of any of the following:
# DEBUG, INFO, WARN, ERROR, CRITICAL, ALERT
# If left empty, will listen for all levels
levels=CRITICAL,ALERT
smtp_host=smtp.gmail.com
smtp_port=465
smtp_user=mercury@gmail.com
smtp_pass=*********
smtp_use_ssl=1
smtp_use_tls=0
to_addr=cassandra_admin@acme.com
from_addr=mercury@gmail.com
subject=OpsCenter Event
```

To enable email alerts to multiple addresses:

Create a different email `conf` file with settings for each email address. All `conf` files are loaded so you can name them `email1.conf`, `email2.conf`, and so on.

Enabling alerts posted to a URL

OpsCenter can post alerts to a URL if you provide a correctly formatted POST script. For example, a simple PHP script containing `print_r($_POST);` should be sufficient for getting started.
To enable URL posting on the OpsCenter side, you must edit `posturl.conf` and provide a path to your script. This file is located in the following directories:

- **Package installations**: `/etc/opscenter/event-plugins`
- **Binary tarball installations** (Linux and Mac OSX): `<install_location/opscenter/conf/event-plugins`
- **Windows** installations: 
  `Program Files (x86)\DataStax Community\opscenter\conf\event-plugins`

**To enable URL posting:**

1. Make sure your web server and posting script are configured to receive alerts.
2. On the OpsCenter daemon host, open `posturl.conf` for editing.
3. Set `enabled` to `1`.
4. For `url`, provide a valid path to your posting script. For example: `url=http://50.1.1.11/postOPSCevents.php`.
5. Optionally, select the desired logging level. The default is to listen for all levels of events.
6. Save `posturl.conf` and restart the OpsCenter daemon.

In a system with posting enabled for critical and alert-level events, `posturl.conf` looks like:

```
[posturl]
enabled=1
url=http://50.1.1.11/postOPSCevents.php
# levels can be comma delimited list of any of the following:
# DEBUG, INFO, WARN, ERROR, CRITICAL, ALERT
# If left empty, will listen for all levels
levels=C R I T I C A L, Alert
```

You can set preferences to specify how the posting is handled on the receiving side.

**To verify that events are posting correctly:**

1. Post events to a file such as `/tmp/events` on the web server host.
2. Create a script. For example, create a PHP script `http://50.1.1.11/postOPSCevents.php`:
   ```php
   <?php
   file_put_contents('/tmp/events', print_r($_POST,true), FILE_APPEND);
   ?>
   ```
3. Deploy the script. You might need to restart the web server.
4. Launch a logged event, such as an OpsCenter restart or garbage compaction from Dashboard > Cluster > List View. Output to /tmp looks something like this:

   Array
   (
       [api_source_ip] => 67.169.50.240
       [target_node] => None
       [event_source] => OpsCenter
       [user] => None
       [time] => 1311025598851602
       [action] => 20
       [message] => Garbage Collecting node 50.1.1.24
       [level] => 1
       [source_node] => 50.1.1.24
       [level_str] => INFO
   )

**Configuring data collection and expiration**

OpsCenter collects system and column family metrics data for each node in your cluster. OpsCenter creates its own keyspace within a cluster for storing collected metrics. Metrics data is collected at regular intervals and stored within your cluster in a keyspace called `OpsCenter`. The column families containing metric data continue to grow. You can configure how long you want to keep historical metrics. Data expires after configurable time periods.

**Estimating the amount of data generated**

The following table provides guidance for estimating the amount of metrics data generated:

<table>
<thead>
<tr>
<th>Number of days</th>
<th>Number of column families monitored</th>
<th>MB per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>5</td>
<td>200</td>
</tr>
<tr>
<td>31</td>
<td>10</td>
<td>300</td>
</tr>
<tr>
<td>31</td>
<td>20</td>
<td>500</td>
</tr>
<tr>
<td>365</td>
<td>5</td>
<td>250</td>
</tr>
<tr>
<td>365</td>
<td>10</td>
<td>380</td>
</tr>
<tr>
<td>365</td>
<td>20</td>
<td>630</td>
</tr>
</tbody>
</table>

The default upper limit of data collected is 365 days.

**Controlling data collection**

To help control consumption of disk space, OpsCenter provides two ways to limit the growth of OpsCenter performance data:

- By excluding specified keyspaces and column families from performance data collection
- By shortening the time period after which performance data automatically expires

**Excluding keyspaces and column families**

By default, OpsCenter does not collect performance data for its own keyspace or the Cassandra system keyspace. You can manually add any other keyspaces or column families that you do not want to monitor in the `[cassandra_metrics]` section of the configuration file.

For example, to prevent data collection for the keyspace `test` as well as the column family `Keyspace1.Standard1`, uncomment and edit the following values in the OpsCenter cluster configuration file (`<cluster_specific>.conf`):
Configure OpsCenter

[cassandra_metrics]
ignored_keyspaces = system, OpsCenter, test
ignored_column_families = Keyspace1.Standard1

Column families are specified in the format:
<keyspace_name>.<column_family_name>

Changing performance data expiration times
Performance data stored in OpsCenter expires after configurable time periods. The default values are designed to provide efficient compaction and eventual deletion of the data, with faster expiration times for the more granular, larger-volume data rollups.

- One-minute rollups (1min_ttl) expire after after one week, or 604800 seconds.
- Five-minute rollups (5min_ttl) expire after four weeks, or 2419200 seconds.
- Two-hour rollups (2hr_ttl) expire after one year, or 31536000 seconds.

To change expiration time period:

1. Uncomment and edit the time-to-live (ttl) values in opscenerd.conf.
2. Restart OpsCenter.

In the following example, the one-minute and five-minute rollups are set to expire twice as fast as the defaults, and two-hour rollups are set to be kept indefinitely (expiration is disabled):

1min_ttl = 302400
5min_ttl = 1209600
2hr_ttl = -1

Data collected after restarting OpsCenter expires according to the new setting. The data collected before restarting OpsCenter expires according to the setting in effect when it was collected.

Advanced configuration
Using the OpsCenter console is the most convenient way to configure basic OpsCenter settings. To configure advanced capabilities, you modify configuration files.

The main configuration files for OpsCenter are:

- opscenerd.conf - Configures the properties for the OpsCenter daemon.
- <cluster_specific>.conf - Configures properties for each cluster monitored by OpsCenter. This file is created when you add a cluster to the Opscenter.

OpsCenter configuration properties
The location of the opscenerd.conf file depends on the type of installation:

- Packaged installations: /etc/opscenter/opscenterd.conf
- Binary tarball installations (Linux and Mac OSX): <install_location>/conf/opscenterd.conf
- Windows installations: Program Files (x86)\DataStax Community\opscenter\conf\opscenterd.conf

Note
After changing properties in this file, restart OpsCenter for the changes to take effect.
You configure the following properties in `opscenterd.conf`:

**[agents] ssh_port**

The Secure Shell (SSH) port that listens for agent-OpsCenter communications. Add an `[agents]` section, if one doesn't already exist, to the `opscenterd.conf`. In this section, add the `ssh_port` option and a value for the port number:

```
[agents]
ssh_port = 2222
```

**[webserver] port**

- The HTTP port used for client connections to the OpsCenter web server. Default is 8888.
- Optional HTTPS support.

  To enable HTTPS, remove the comment markers (#) in front of properties prefixed with `ssl` in the `opscenterd.conf` file, as described in *Configuring HTTPS*.

**[webserver] interface**

The interface that the web server uses to listen for client connections. The interface must be an externally accessible IP address or host name.

**[logging] level**

The logging level for OpsCenter. Available levels are (from most to least verbose): TRACE, DEBUG, INFO, WARN, or ERROR. The OpsCenter log file is located in `/var/log/opscenter/opscenterd.log`.

**[stat_reporter] interval**

Reporting to DataStax Support. By default, OpsCenter periodically sends usage metrics about the cluster to DataStax Support. To disable the phone-home functionality, add the following lines to your `opscenterd.conf` file:

```
[stat_reporter]
interval = 0
```

Additional configuration metric collection properties are available in *Metrics Collection Properties*.

**[authentication] passwd_file**

Full path to the file for configuring password authentication for OpsCenter. If this file does not exist, OpsCenter does not verify passwords. To enable password authentication, use the `set_passwd.py` utility to create users and set their password and role. OpsCenter currently has two available roles: *admin* or *user*.

**Cluster configuration properties**

You set OpsCenter configuration properties in the `<cluster_specific.conf>` file. The location of this file depends on the type of installation:

- **Package installations**: `/etc/opscenter/clusters/<cluster_specific>.conf`
- **Binary tarball installations** (Linux and Mac OSX): `<install_location>/conf/clusters/<cluster_specific>.conf`
- **Windows installations**: `Program Files (x86)\DataStax Community\opscenter\conf\clusters\<cluster_specific>.conf`
Note
After changing properties in this file, restart OpsCenter for the changes to take effect.

Cassandra connection properties
The following properties inform OpsCenter about the Real-time (Cassandra), Analytics (Hadoop), and Search (Solr) nodes that it is monitoring:

[jmx] port
The JMX (Java Management Extensions) port of your cluster. In Cassandra versions 0.8 and higher, the JMX port is 7199.

cassandra] seed_hosts
A Cassandra seed node is used to determine the ring topology and obtain gossip information about the nodes in the cluster. This should be the same comma-delimited list of seed nodes as the one configured for your Cassandra or DataStax Enterprise cluster by the seeds property in the cassandra.yaml configuration file.

cassandra] api_port
The Thrift remote procedure call port configured for your cluster. Same as the rpc_port property in the cassandra.yaml configuration file. Default is 9160.

Metrics Collection Properties
The following properties are used to limit the keyspaces and column families for which you collect metrics.

[cassandra_metrics] ignored_keyspaces
A comma-delimited list of Cassandra keyspaces for which you do not want to collect performance metrics. By default, the system and OpsCenter keyspaces are excluded.

[cassandra_metrics] ignored_column_families
A comma-delimited list of Cassandra column families for which you do not want to collect performance metrics. Entries should be in the form of keyspace_name.*columnfamily_name*.

Performance Data Expiration Properties
These properties set the expiration time for data stored in the OpsCenter keyspace. Each time period for rolling up data points into summary views has a separate expiration threshold, or time-to-live (ttl) value expressed in seconds. By default, shorter time periods have lower values that result in more efficient expiration and compaction of the relatively larger volumes of data.

Uncomment these properties to change the default expiration periods for performance data. Properties and default values are:

1min_ttl = 604800
One-minute rollups expire after after one week, or 604800 seconds.

5min_ttl = 2419200
Five-minute rollups expire after four weeks, or 2419200 seconds.
Starting and stopping OpsCenter and its agents

Two-hour rollups expire after one year, or 31536000 seconds.

Starting and stopping OpsCenter and its agents

Packaged installations include startup scripts for running OpsCenter as a service. These options can be used with the service opscenterd command:

```
service opscenterd start|stop|status|restart|force-reload
```

By default, DataStax Enterprise services on Windows start automatically.

Starting OpsCenter

Tarball installation

To start OpsCenter, enter one of the following commands from the `<install_location>`:

- `bin/opscenter` (in the background - default)
- `bin/opscenter -f` (in the foreground)

Packaged installation

To start OpsCenter, enter the following command:

```
sudo service opscenterd start
```

Windows

To start OpsCenter, start the DataStax OpsCenter Service in the Control Panel.

Restarting OpsCenter

Tarball installation

To restart OpsCenter in a tarball installation:

Use the `stop` and `start` procedures to stop and start OpsCenter.

Packaged installation

To restart a packaged installation, enter this command:

```
sudo service opscenterd restart
```

Windows

To restart OpsCenter, restart the DataStax OpsCenter Service in the Control Panel.

Stopping OpsCenter

Tarball installation

To stop OpsCenter, find the OpsCenter Java process ID (PID) and kill the process using its PID number. For example:

```
ps -ef | grep opscenter
sudo kill <pid>
```

Packaged installation

To stop OpsCenter, enter the following command:

```
sudo service opscenterd stop
```
Windows
To stop OpsCenter, stop the DataStax OpsCenter Service in the Control Panel.

Starting the OpsCenter agent
Tarball installation
To start the OpsCenter Agent, enter one of the following commands from the <install_location>.

- `bin/opscenter-agent` (in the background - default)
- `bin/opscenter-agent -f` (in the foreground)

Packaged installation
The OpsCenter Agent starts automatically when you start OpsCenter.

Windows
Start the DataStax OpsCenter Agent Service in the Control Panel.

Restarting the OpsCenter Agent
Packaged installation
If you installed OpsCenter from a package use:

```
sudo service opscenter-agent restart
```

Tarball installation
If you installed DataStax OpsCenter using the binary tarball or if you installed the agent using the agent tar installer (agent.tar.gz):

1. Stop the OpsCenter agent.

```
ps -ef | grep opscenter-agent
kill <pid>
```

2. Restart the OpsCenter agent from the <install_location>.

```
bin/opscenter-agent (in the background - default)
bin/opscenter-agent -f (in the foreground)
```

Windows
Restart the DataStax OpsCenter Agent Service in the Control Panel.

Using OpsCenter
OpsCenter is a graphical user interface for monitoring and administering all nodes in a Cassandra cluster from one centralized console. Opscenter 2.1.3 supports Cassandra 1.2 for monitoring and administering thousands of Cassandra column families efficiently.

Introduction to OpsCenter
At the top of every functional area of OpsCenter, you can access these functions:

- **Add a cluster**, the dialog for adding multiple clusters to OpsCenter to monitor. Available in OpsCenter Enterprise Edition only.
- **Feedback**, an online form that sends your evaluation of OpsCenter or any comments to us.
Using OpsCenter

- **Report**, information about clusters that OpsCenter manages in PDF format.

OpsCenter is divided into these main functional areas:

- **Overview** - Survey each cluster's Dashboard in this condensed view. Displayed when multiple clusters are present.
- **Dashboard** - View graphs of the most commonly watched Cassandra performance metrics.
- **Cluster views** - See your cluster from different perspectives and perform certain maintenance operations on cluster nodes.
- **Cluster administration** - Available in OpsCenter Enterprise Edition only. Add, modify, or remove a cluster from OpsCenter.
- **Performance** - Monitor a number of Cassandra cluster performance metrics. Real-time and historical performance metrics are available at different granularities: cluster-wide, per node, or per column family.
- **Alerts** - Available in OpsCenter Enterprise Edition only. Configure alert thresholds for a number of Cassandra cluster-wide, column family, and operating system metrics.
- **Data Backups** - Available in OpsCenter Enterprise Edition only. Visually take, schedule, and manage backups across all registered clusters.
- **Data Modeling** - Define new keyspaces and column families.
- **Data Explorer** - Browse through column family data.
- **Event Log** - View the most recent OpsCenter log events, such as OpsCenter start-up and shutdown.

**Cluster and node administration**

In the Cluster area of OpsCenter, you select different views of the nodes comprising your Cassandra cluster, and then, perform node management and **cluster rebalancing** operations.

OpsCenter Enterprise Edition 2.1.3 can monitor and administer Cassandra 1.2 as described in these procedures unless you enable virtual nodes. When you enable virtual nodes, OpsCenter chooses a single token for each node for operations, such as collecting metrics. Attempting to move nodes, rebalance nodes, and perform other tasks involving token ranges is not supported.

**Node management operations**

Each of the cluster views—ring, physical, and list—has an Actions button.

- In the Ring View or Physical View of a cluster, click the graphic representation of the node. The Actions button appears in the dialog.
- In the List View of a cluster, select a node. If necessary, expand the List View window to see the Actions button on the right side.

Click the Actions button to access a drop-down list of node management operations. To avoid impacting cluster performance, perform actions that move data between nodes at low-usage times. Examples of such actions are move, decommission, and repair.

This table describes actions on the drop-down list:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Metrics</td>
<td>Redirects you to the Performance area of OpsCenter where you can select metrics graphs and configure performance views for the selected node.</td>
</tr>
<tr>
<td>View Replication</td>
<td>Shows the replication relationships between the selected node and other nodes in the cluster. Visible in Ring View and Physical View.</td>
</tr>
<tr>
<td>Cleanup</td>
<td>Removes rows that the node is no longer responsible for. This is usually done after changing the partitioner tokens or the replication options for a cluster.</td>
</tr>
</tbody>
</table>
Compact | Performs a major compaction, which is not a recommended procedure in most Cassandra clusters.
Flush | Causes the recent writes currently stored in memory (memtables) to be flushed to disk as persistent SSTables.
Repair | Makes a node consistent with its replicas by doing an in-memory comparison of all the rows of a column family, and resolving any discrepancies between replicas by updating outdated rows with the current data.
Perform GC | Forces the Java Virtual Machine (JVM) on the selected node to perform a garbage collection (GC). This reclaims physical disk space occupied by obsolete SSTables, such as those that have been truncated or compacted (merged) into new SSTables.
Decommission | Removes a node from the cluster and streams its data to neighboring replicas.
Drain | Causes the recent writes currently stored in memory (memtables) to be flushed to disk as persistent SSTables, and then makes the node read-only (the node will stop accepting new writes). This is usually done when upgrading a node.
Move | Changes the partitioner token assignment for the node, thus changing the range of data that the node is responsible for.

Rebalancing a cluster

Cluster rebalancing is a process that makes sure each node in a Cassandra cluster is managing an equal amount of data. Currently, OpsCenter only supports rebalancing on clusters using the random partitioner or murmur 3 partitioner. Ordered partitioners are not supported. When using the random partitioner or murmur 3 partitioner, a rebalance is usually required only when you have changed the cluster topology in some way, such as adding or removing nodes or changing the replica placement strategy.

A cluster is considered balanced when each node is responsible for an equal range of data. This is done by evaluating the partitioner tokens assigned to each node to make sure that the data ranges each node is responsible for are even. Even though a cluster is considered balanced, it is still possible that one or more nodes have more data than the others. This is because the size of the rows is not taken into account, only the number of rows managed by each node.

To rebalance a cluster:

1. In the Cluster section of OpsCenter, select Ring View, Physical View or List View. This shows a view of your cluster.
2. Click the Rebalance Cluster button. This causes OpsCenter to check if the token ranges are evenly distributed across the nodes in your cluster. If your cluster is already balanced, then there is nothing for OpsCenter to do.
3. If the cluster does require rebalancing, OpsCenter performs the following steps:
   a. Calculates appropriate token ranges for each node and identify the nodes that need to move.
   b. Makes sure that there is the appropriate free space to perform the rebalancing.
   c. Moves nodes, one node at a time so as to lessen the impact on the cluster workloads. A move operation involves changing the partitioner token assignment for the node, thus changing the range of data that the node is responsible for. A move will stream data from other nodes.
   d. After a move is complete on a node, runs cleanup. A cleanup operation removes rows that the node is no longer responsible for.
4. If you cancel a rebalance operation before all nodes are moved, you can resume it at a later time by clicking the Rebalance Cluster button again.

Cluster administration

You can add, edit, or delete a cluster from OpsCenter and generate a PDF report about the cluster. OpsCenter Enterprise Edition 2.1.3 can monitor and administer Cassandra 1.2 as described in these procedures unless you enable
virtual nodes. When you enable virtual nodes, OpsCenter chooses a single token for each node for operations, such as collecting metrics. Attempting to move nodes, rebalance nodes, and perform other tasks involving token ranges is not supported.

**Prerequisites for administering a cluster**

You must have a running and properly configured Cassandra cluster, as described in the documentation. OpsCenter 2.1.3 and later supports:

- Cassandra 1.2 clusters with limitations if you enable virtual nodes
- Cassandra 0.8 - 1.1.x clusters
- DataStax Enterprise 2.2.x and earlier clusters, including multiple data center clusters

**Adding a cluster to OpsCenter**

To add a cluster to OpsCenter:

1. Click Add a Cluster.
2. Enter the hostname or IP address for the nodes comprising the cluster. For example:
   - ec2-123-45-6-789.us-west-1.compute.amazonaws.com
   - ec2-234-56-7-890.us-west-1.compute.amazonaws.com
3. If you are not using the default JMX or Thrift ports, enter the appropriate port numbers.
4. If required, click Add Credentials and enter the username and password for JMX or Thrift ports.
5. Click Add Cluster.

**Modifying a cluster setting**

To modify a cluster setting:

1. Click Edit Cluster.
2. Change the IP addresses of cluster nodes.
3. Change JMX and Thrift listen port numbers. Click Add credentials if the ports require authentication.
4. Click Save Cluster.

**Removing a cluster from OpsCenter**

To remove a cluster

This removes the cluster from OpsCenter, it does not delete the cluster.

1. Click Edit Cluster.
2. Click Delete Cluster.

**Generating a report**

To generate a PDF report about the cluster being monitored, click Report at the top of the OpsCenter interface. The report shows the version of OpsCenter, number of clusters and nodes being monitoring, gigabytes of storage used, name of the cluster, and information about nodes in the cluster. The node information includes:

- Node name and IP address
- Cassandra software version
- DataStax software version
Using OpsCenter

- Memory usage
- Operating system running on the node

You can save or print the PDF report.

**OpsCenter performance metrics**

In the Performance area of OpsCenter, you monitor a number of performance metrics about a Cassandra cluster. Real-time and historical performance metrics are available at different granularities: cluster-wide, per node, or per column family. OpsCenter 2.1 and later has been optimized to handle thousands of column families efficiently.

This section contains the following topics:

**Using performance metrics**

Select **Performance** in the OpsCenter Console to view these types of metrics:

- Cluster Performance Metrics
- Pending Task Metrics
- Column Family Metrics

When you add a graph, you choose the Metric and the source that OpsCenter uses to collect the data for the graph:

- Cluster wide
- All nodes
- The node running Opscenter

Several commonly-used performance metrics graphs are displayed initially. Data appears in the graphs after you set alerts.

You can save, delete, and choose the default view of graphs. Click the link to save presets at the top of the Performance area. The Save, Delete, and Make Default menu options are available after saving more than one view.

**Cluster performance metrics**

Cluster metrics are aggregated across all nodes in the cluster. Cluster metrics are a good way to monitor cluster performance at a high level. OpsCenter tracks a number of cluster-wide metrics for read performance, write performance, memory and capacity.

Watching for variations in cluster performance can signal potential performance issues that may require further investigation. For general performance monitoring, watching for spikes in read and write latency, along with an accumulation of pending operations can signal issues that may require further investigation. Drilling down on high-demand column families can further pinpoint the source of performance issues with your application.

**Write Requests**

The number of write requests per second. Monitoring the number of requests over a given time period can give you an idea of system write workload and usage patterns.

**Write Request Latency**

The response time (in microseconds) for successful write requests. The time period starts when a node receives a client write request, and ends when the node responds back to the client. Optimal or acceptable levels of write latency vary widely according to your hardware, your network, and the nature of your write load. For example, the performance for a write load consisting largely of granular data at low consistency levels would be evaluated differently from a load of large strings written at high consistency levels.
Read Requests
The number of read requests per second. Monitoring the number of requests over a given time period can give you an idea of system read workload and usage patterns.

Read Request Latency
The response time (in microseconds) for successful read requests. The time period starts when a node receives a client read request, and ends when the node responds back to the client. Optimal or acceptable levels of read latency vary widely according to your hardware, your network, and the nature of your application read patterns. For example, the use of secondary indexes, the size of the data being requested, and the consistency level required by the client can all impact read latency. An increase in read latency can signal I/O contention. Reads can slow down when rows are fragmented across many SSTables and compaction cannot keep up with the write load.

Cassandra JVM Memory Usage
The average amount of Java heap memory (in megabytes) being used by Cassandra processes. Cassandra opens the JVM with a heap size that is half of available system memory by default, which still allows an optimal amount of memory remaining for the OS disk cache. You may need to increase the amount of heap memory if you have increased column family memtable or cache sizes and are getting out-of-memory errors. If you monitor Cassandra Java processes with an OS tool such as top, you may notice the total amount of memory in use exceeds the maximum amount specified for the Java heap. This is because Java allocates memory for other things besides the heap. It is not unusual for the total memory consumption of the JVM to exceed the maximum value of heap memory.

JVM CMS Collection Count
The number of concurrent mark-sweep (CMS) garbage collections performed by the JVM per second. These are large, resource-intensive collections. Typically, the collections occur every 5 to 30 seconds.

JVM CMS Collection Time
The time spent collecting CMS garbage in milliseconds per second (ms/sec).

Note
A ms/sec unit defines the number of milliseconds for garbage collection for each second that passes. For example, the percentage of time spent on garbage collection in one millisecond (.001 sec) is 0.1%.

JVM ParNew Collection Count
The number of parallel new-generation garbage collections performed by the JVM per second. These are small and not resource intensive. Normally, these collections occur several times per second under load.

JVM ParNew Collection Time
The time spent performing ParNew garbage collections in ms/sec. The rest of the JVM is paused during ParNew garbage collection. A serious performance hit can result from spending a significant fraction of time on ParNew collections.

Data Size
The size of column family data (in gigabytes) that has been loaded/inserted into Cassandra, including any storage overhead and system metadata. DataStax recommends that data size not exceed 70 percent of total disk capacity to allow free space for maintenance operations such as compaction and repair.

Total Bytes Compacted
Using OpsCenter

The number of sstable data compacted in bytes per second.

**Total Compactions**
The number of compactions (minor or major) performed per second.

**Pending task metrics**
Pending task metrics track requests that have been received by a node, but are waiting to be processed. An accumulation of pending tasks on a node can indicate a potential bottleneck in performance and should be investigated.

Cassandra maintains distinct thread pools for different stages of execution. Each of these thread pools provide granular statistics on the number of pending tasks for that particular process. If you see pending tasks accumulating, it is indicative of a cluster that is not keeping up with the workload. Essentially, pending tasks mean that things are backing up, which is usually caused by a lack of (or failure of) cluster resources such as disk bandwidth, network bandwidth or memory.

**Pending Task Metrics for Writes**
Pending tasks for the following metrics indicate that write requests are arriving faster than they can be handled.

**Flushes Pending**
The flush process flushes memtables to disk as SSTables. This metric shows the number of memtables queued for the flush process. The optimal number of pending flushes is 0 (or at most a very small number). A value greater than 0 indicates either I/O contention or degrading disk performance (see disk metrics such as disk latency, disk throughput, and disk utilization for indications of disk health).

**Flush Sorter Tasks Pending**
The flush sorter process performs the first step in the overall process of flushing memtables to disk as SSTables.

**Memtable Post Flushers Pending**
The memtable post flush process performs the final step in the overall process of flushing memtables to disk as SSTables.

**Write Requests Pending**
The number of write requests that have arrived into the cluster but are waiting to be handled. During low or moderate write load, you should see 0 pending write operations (or at most a very low number). A continuous high number of pending writes signals a need for more capacity in your cluster or to investigate disk I/O contention.

**Replicate on Write Tasks Pending**
When an insert or update to a row is written, the affected row is replicated to all other nodes that manage a replica for that row. This is called the ReplicateOnWriteStage. This metric tracks the pending tasks related to this stage of the write process. During low or moderate write load, you should see 0 pending replicate on write tasks (or at most a very low number). A continuous high number signals a need to investigate disk I/O or network contention problems.

**Pending Task Metrics for Reads**
Pending tasks for the following metrics indicate I/O contention, and can manifest in degrading read performance.

**Read Requests Pending**
The number of read requests that have arrived into the cluster but are waiting to be handled. During low or moderate read load, you should see 0 pending read operations (or at most a very low number). A continuous high number of
pending reads signals a need for more capacity in your cluster or to investigate disk I/O contention. Pending reads can also indicate an application design that is not accessing data in the most efficient way possible.

**Read Repair Tasks Pending**

The number of read repair operations that are queued and waiting for system resources in order to run. The optimal number of pending read repairs is 0 (or at most a very small number). A value greater than 0 indicates that read repair operations are in I/O contention with other operations. If this graph shows high values for pending tasks, this may suggest the need to run a node repair to make nodes consistent. Or, for column families where your requirements can tolerate a certain degree of stale data, you can lower the value of the column family parameter read_repair_chance.

**Compactions Pending**

An upper bound of the number of compactions that are queued and waiting for system resources in order to run. This is a worst-case estimate. The compactions pending metric is often misleading. An unrealistic, high reading often occurs. The optimal number of pending compactions is 0 (or at most a very small number). A value greater than 0 indicates that read operations are in I/O contention with compaction operations, which usually manifests itself as declining read performance. This is usually caused by applications that perform frequent small writes in combination with a steady stream of reads. If a node or cluster frequently displays pending compactions, that is an indicator that you may need to increase I/O capacity by adding nodes to the cluster. You can also try to reduce I/O contention by reducing the number of insert/update requests (have your application batch writes for example), or reduce the number of SSTables created by increasing the memtable size and flush frequency on your column families.

**Pending Task Metrics for Cluster Operations**

Pending tasks for the following metrics indicate a backup of cluster operational processes such as those maintaining node consistency, system schemas, fault detection, and inter-node communications. Pending tasks for resource-intensive operations (such as repair, bootstrap or decommission) are normal and expected while that operation is in progress, but should continue decreasing at a steady rate in a healthy cluster.

**Manual Repair Tasks Pending**

The number of operations still to be completed when you run anti-entropy repair on a node. It will only show values greater than 0 when a repair is in progress. Repair is a resource-intensive operation that is executed in stages: comparing data between replicas, sending changed rows to the replicas that need to be made consistent, deleting expired tombstones, and rebuilding row indexes and bloom filters. Tracking the state of this metric can help you determine the progress of a repair operation. It is not unusual to see a large number of pending tasks when a repair is running, but you should see the number of tasks progressively decreasing.

**Gossip Tasks Pending**

Cassandra uses a protocol called gossip to discover location and state information about the other nodes participating in a Cassandra cluster. In Cassandra, the gossip process runs once per second on each node and exchanges state messages with up to three other nodes in the cluster. Gossip tasks pending shows the number of gossip messages and acknowledgments queued and waiting to be sent or received. The optimal number of pending gossip tasks is 0 (or at most a very small number). A value greater than 0 indicates possible network problems (see network traffic for indications of network health).

**Hinted Handoff Pending**

While a node is offline, other nodes in the cluster will save hints about rows that were updated during the time the node was unavailable. When a node comes back online, its corresponding replicas will begin streaming the missed writes to the node to catch it up. The hinted handoff pending metric tracks the number of hints that are queued and waiting to be delivered once a failed node is back online again. High numbers of pending hints are commonly seen when a node is brought back online after some down time. Viewing this metric can help you determine when the recovering node has been made consistent again. Hinted handoff is an optional feature of Cassandra. Hints are saved for a configurable period of time (an hour by default) before they are dropped. This prevents a large accumulation of hints caused by
Using OpsCenter

extended node outages.

**Internal Responses Pending**
The number of pending tasks from various internal tasks such as nodes joining and leaving the cluster.

**Migrations Pending**
The number of pending tasks from system methods that have modified the schema. Schema updates have to be propagated to all nodes, so pending tasks for this metric can manifest in schema disagreement errors.

**Misc. Tasks Pending**
The number of pending tasks from other miscellaneous operations that are not ran frequently.

**Request Responses Pending**
The progress of rows of data being streamed from the *receiving* node. Streaming of data between nodes happens during operations such as bootstrap and decommission when one node sends large numbers of rows to another node.

**Streams Pending**
The progress of rows of data being streamed from the *sending* node. Streaming of data between nodes happens during operations such as bootstrap and decommission when one node sends large numbers of rows to another node.

**Column family performance metrics**
Column family metrics allow you to drill down and locate specific areas of your application workloads that are the source of performance issues. If you notice a performance trend at the OS or cluster level, viewing column family metrics can provide a more granular level of detail.

The metrics for KeyCache Hits, RowCache Hits and SSTable Size can only be viewed on a single column family at a time. Otherwise, all column family metrics are available for specific column families as well as for all column families on a node.

In addition to monitoring read latency, write latency and load on a column family, you should also monitor the hit rates on the key and row caches for column families that rely on caching for performance. The more requests that are served from the cache, the better response times will be.

OpsCenter 2.1 and later has been optimized to handle thousands of column families efficiently. If a column family experiences a dramatic dip in performance, check the Pending Tasks metric for a back-up in queued operations.

Viewing SSTable Size and SSTable Count for a specific column family (or counts for all families) can help with compaction tuning.

**CF: Local Writes**
The write load on a column family measured in requests per second. This metric includes all writes to a given column family, including write requests forwarded from other nodes. This metric can be useful for tracking usage patterns of your application.

**CF: Local Write Latency**
The response time in microseconds for successful write requests on a column family. The time period starts when nodes receive a write request, and ends when nodes respond. Optimal or acceptable levels of write latency vary widely according to your hardware, your network, and the nature of your write load. For example, the performance for a write load consisting largely of granular data at low consistency levels would be evaluated differently from a load of large strings written at high consistency levels.
**CF: Local Reads**
The read load on a column family measured in requests per second. This metric includes all reads to a given column family, including read requests forwarded from other nodes. This metric can be useful for tracking usage patterns of your application.

**CF: Local Read Latency**
The response time in microseconds for successful reads on a column family. The time period starts when a node receives a read request, and ends when the node responds. Optimal or acceptable levels of read latency vary widely according to your hardware, your network, and the nature of your application read patterns. For example, the use of secondary indexes, the size of the data being requested, and the consistency level required by the client can all impact read latency. An increase in read latency can signal I/O contention. Reads can slow down when rows are fragmented across many SSTables and compaction cannot keep up with the write load.

**CF: KeyCache Requests**
The total number of read requests on the row key cache.

**CF: KeyCache Hits**
The number of read requests that resulted in the requested row key being found in the key cache.

**CF: KeyCache Hit Rate**
The percentage of cache requests that resulted in a cache hit that indicates the effectiveness of the key cache for a given column family. The key cache is used to find the exact location of a row on disk. If a row is not in the key cache, a read operation will populate the key cache after accessing the row on disk so subsequent reads of the row can benefit. Each hit on a key cache can save one disk seek per SSTable. If the hits line tracks close to the requests line, the column family is benefiting from caching. If the hits fall far below the request rate, this suggests that you could take actions to improve the performance benefit provided by the key cache, such as adjusting the number of keys cached.

**CF: RowCache Requests**
The total number of read requests on the row cache. This metric is only meaningful for column families with row caching configured (it is not enabled by default).

**CF: RowCache Hits**
The number of read requests that resulted in the read being satisfied from the row cache. This metric is only meaningful for column families with row caching configured (it is not enabled by default).

**CF: Row Cache Hit Rate**
The percentage of cache requests that resulted in a cache hit that indicates the effectiveness of the row cache for a given column family. This metric is only meaningful for column families with row caching configured (it is not enabled by default). The graph tracks the number of read requests in relationship to the number of row cache hits. If the hits line tracks close to the requests line, the column family is benefiting from caching. If the hits fall far below the request rate, this suggests that you could take actions to improve the performance benefit provided by the row cache, such as adjusting the number of rows cached or modifying your data model to isolate high-demand rows.

**CF: SSTable Size**
The current size of the SSTables for a column family. It is expected that SSTable size will grow over time with your write load, as compaction processes continue doubling the size of SSTables. Using this metric together with SSTable count, you can monitor the current state of compaction for a given column family. Viewing these patterns can be helpful if you are considering reconfiguring compaction settings to mitigate I/O contention.
**CF: SSTable Count**

The current number of SSTables for a column family. When column family memtables are persisted to disk as SSTables, this metric increases to the configured maximum before the compaction cycle is repeated. Using this metric together with SSTable size, you can monitor the current state of compaction for a given column family. Viewing these patterns can be helpful if you are considering reconfiguring compaction settings to mitigate I/O contention.

**CF: Pending Reads/Writes**

The number of pending reads and writes on a column family. Pending operations are an indication that Cassandra is not keeping up with the workload. A value of zero indicates healthy throughput. If out-of-memory events become an issue in your Cassandra cluster, it may help to check cluster-wide pending tasks for operations that may be clogging throughput. Bloom filters are used to avoid going to disk to try to read rows that don't actually exist.

**CF: Bloom Filter Space Used**

The size of the bloom filter files on disk. This grows based on the number of rows in a column family and is tunable through the per-CF attribute, `bloom_filter_fp_chance`; increasing the value of this attribute shrinks the bloom filters at the expense of a higher number of false positives. Cassandra reads the bloom filter files and stores them on the heap, so large bloom filters can be expensive in terms of memory consumption.

*Note*

Bloom filters are used to avoid going to disk to try to read rows that don't actually exist.

**CF: Bloom Filter False Positives**

The number of false positives, which occur when the bloom filter said the row existed, but it actually did not exist in absolute numbers.

**CF: Bloom Filter False Positive Ratio**

The fraction of all bloom filter checks resulting in a false positive. This should normally be at or below .01. A higher reading indicates that the bloom filter is likely too small.

**System performance metrics**

As with any database system, Cassandra performance greatly depends on underlying systems on which it is running. Tracking operating system metrics on your Cassandra nodes to watch for disk I/O, network, memory and CPU utilization trends can help you identify and troubleshoot hardware-related performance problems.

Monitoring Cassandra nodes for increasing disk and CPU utilization can help identify and remedy issues before performance degrades to unacceptable levels. The graphs in OpsCenter provide a quick way to view variations in OS metrics at a glance, and drill-down for specific data points. Especially in systems with heavy write loads, monitoring disk space is also important. It allows for advanced expansion planning while there is still adequate capacity to handle expansion and rebalancing operations.

**Memory**

Shows memory usage metrics in megabytes.

- **Linux** -- Shows how much total system memory is currently used, cached, buffered or free.
- **Windows** -- Shows the available physical memory, the cached operating system code, and the allocated pool-paged-resident and pool-nonpaged memory.
- **Mac OSX** -- Shows free and used system memory.
**CPU**

Shows average percentages for CPU utilization metrics, which is the percentage of time the CPU was idle subtracted from 100 percent. CPU metrics can be useful for determining the origin of CPU performance reduction.

Linux -- Shows how much time the CPU devotes to system and user processes, to tasks stolen by virtual operating systems, to waiting for I/O to complete, and to processing nice tasks. High percentages of nice may indicate that other processes are crowding out Cassandra processes, while high percentages of iowait may indicate I/O contention. On fully virtualized environments like Amazon EC2, a Cassandra cluster under load may show high steal values while other virtual processors use the available system resources.

Windows and Mac OSX -- Shows how much time the CPU spends on user processes and system processes.

**Load**

The amount of work that a computer system performs. An idle computer has a load number of 0 and each process using or waiting for CPU time increments the load number by 1. Any value above one indicates that the machine was temporarily overloaded and some processes were required to wait. Shows minimum, average, and maximum OS load expressed as an integer.

**Disk Usage (GB)**

Tracks growth or reduction in the amount of available disk space used. If this metric indicates a growth trend leading to high or total disk space usage, consider strategies to relieve it, such as adding capacity to the cluster. DataStax recommends leaving 30-50% free disk space for optimal repair and compaction operations.

**Disk Usage (%)**

The percentage of disk space that is being used by Cassandra at a given time. When Cassandra is reading and writing heavily from disk, or building SSTables as the final product of compaction processes, disk usage values may be temporarily higher than expected.

**Disk Throughput**

The average disk throughput for read and write operations, measured in megabytes per second. Exceptionally high disk throughput values may indicate I/O contention. This is typically caused by numerous compaction processes competing with read operations. Reducing the frequency of memtable flushing can relieve I/O contention.

**Disk Rates**

Linux and Windows -- Averaged disk speed for read and write operations.

Mac OSX -- Not supported.

**Disk Latency**

Linux and Windows -- Measures the average time consumed by disk seeks in milliseconds. Disk latency is among the higher-level metrics that may be useful to monitor on an ongoing basis by keeping this graph posted on your OpsCenter performance console. Consistently high disk latency may be a signal to investigate causes, such as I/O contention from compactions or read/write loads that call for expanded capacity.

Mac OSX -- Not supported.

**Disk Request Size**

Linux and Windows -- The average size in sectors of requests issued to the disk.

Mac OSX -- Not supported.
Using OpsCenter

**Disk Queue Size**
Linux and Windows -- The average number of requests queued due to disk latency issues.
Mac OS X -- Not supported.

**Disk Utilization**
Linux and Windows -- The percentage of CPU time consumed by disk I/O.
Mac OS X -- Not supported.

**Network Traffic**
The speed at which data is received and sent across the network, measured in kilobytes per second.

**Definitions of alert metrics**
From the Alerts area of OpsCenter Enterprise Edition, you can configure alert thresholds for a number of Cassandra cluster-wide, column family, and operating system metrics. This proactive monitoring feature is available only in OpsCenter Enterprise Edition.

**Commonly watched alert metrics**
OpsCenter provides the capability to configure alerts for the following most commonly watched Cassandra and system metrics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Down</td>
<td>When a node is not responding to requests, it is marked as down.</td>
</tr>
<tr>
<td>Write Requests</td>
<td>The number of write requests per second. Monitoring the number of writes over a given time period can give you an idea of system write workload and usage patterns.</td>
</tr>
<tr>
<td>Write Request Latency</td>
<td>The response time (in microseconds) for successful write operations. The time period starts when a node receives a client write request, and ends when the node responds back to the client.</td>
</tr>
<tr>
<td>Read Requests</td>
<td>The number of read requests per second. Monitoring the number of reads over a given time period can give you an idea of system read workload and usage patterns.</td>
</tr>
<tr>
<td>Read Request Latency</td>
<td>The response time (in microseconds) for successful read operations. The time period starts when a node receives a client read request, and ends when the node responds back to the client.</td>
</tr>
<tr>
<td>CPU Usage</td>
<td>The percentage of time that the CPU was busy, which is calculated by subtracting the percentage of time the CPU was idle from 100 percent.</td>
</tr>
<tr>
<td>Load</td>
<td>Load is a measure of the amount of work that a computer system performs. An idle computer has a load number of 0 and each process using or waiting for CPU time increments the load number by 1.</td>
</tr>
</tbody>
</table>

**Advanced Cassandra alert metrics**
OpsCenter provides the ability to configure alerts for the following Cassandra metrics. These metrics are aggregated across all nodes in the cluster.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
</table>

39
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap Max</td>
<td>The maximum amount of shared memory allocated to the JVM heap for Cassandra processes.</td>
</tr>
<tr>
<td>Heap Used</td>
<td>The amount of shared memory in use by the JVM heap for Cassandra processes.</td>
</tr>
<tr>
<td>JVM CMS Collection Count</td>
<td>The number of concurrent mark-sweep (CMS) garbage collections performed by the JVM per second.</td>
</tr>
<tr>
<td>JVM ParNew Collection Count</td>
<td>The number of parallel new-generation garbage collections performed by the JVM per second.</td>
</tr>
<tr>
<td>JVM CMS Collection Time</td>
<td>The time spent collecting CMS garbage in milliseconds per second (ms/sec).</td>
</tr>
<tr>
<td>JVM ParNew Collection Time</td>
<td>The time spent performing ParNew garbage collections in ms/sec.</td>
</tr>
<tr>
<td>Data Size</td>
<td>The size of column family data (in gigabytes) that has been loaded/inserted into Cassandra, including any storage overhead and system metadata.</td>
</tr>
<tr>
<td>Compactions Pending</td>
<td>The number of compaction operations that are queued and waiting for system resources in order to run. The optimal number of pending compactions is 0 (or at most a very small number). A value greater than 0 indicates that read operations are in I/O contention with compaction operations, which usually manifests itself as declining read performance.</td>
</tr>
<tr>
<td>Total Bytes Compacted</td>
<td>The number of sstable data compacted in bytes per second.</td>
</tr>
<tr>
<td>Total Compactions</td>
<td>The number of compactions (minor or major) performed per second.</td>
</tr>
<tr>
<td>Flush Sorter Tasks Pending</td>
<td>The flush sorter process performs the first step in the overall process of flushing memtables to disk as SSTables. The optimal number of pending flushes is 0 (or at most a very small number).</td>
</tr>
<tr>
<td>Flushes Pending</td>
<td>The flush process flushes memtables to disk as SSTables. This metric shows the number of memtables queued for the flush process. The optimal number of pending flushes is 0 (or at most a very small number).</td>
</tr>
<tr>
<td>Gossip Tasks Pending</td>
<td>Cassandra uses a protocol called gossip to discover location and state information about the other nodes participating in a Cassandra cluster. In Cassandra, the gossip process runs once per second on each node and exchanges state messages with up to three other nodes in the cluster. Gossip tasks pending shows the number of gossip messages and acknowledgments queued and waiting to be sent or received. The optimal number of pending gossip tasks is 0 (or at most a very small number).</td>
</tr>
<tr>
<td>Hinted Handoff Pending</td>
<td>While a node is offline, other nodes in the cluster will save hints about rows that were updated during the time the node was unavailable. When a node comes back online, its corresponding replicas will begin streaming the missed writes to the node to catch it up. The hinted handoff pending metric tracks the number of hints that are queued and waiting to be delivered once a failed node is back online again. High numbers of pending hints are commonly seen when a node is brought back online after some down time. Viewing this metric can help you determine when the recovering node has been made consistent again.</td>
</tr>
<tr>
<td>Internal Responses Pending</td>
<td>The number of pending tasks from various internal tasks such as nodes joining and leaving the cluster.</td>
</tr>
<tr>
<td>Manual Repair Tasks Pending</td>
<td>The number of operations still to be completed when you run anti-entropy repair on a node. It will only show values greater than 0 when a repair is in progress. It is not unusual to see a large number of pending tasks when a repair is running, but you should see the number of tasks progressively decreasing.</td>
</tr>
<tr>
<td>Memtable Post Flushers Pending</td>
<td>The memtable post flush process performs the final step in the overall process of flushing memtables to disk as SSTables. The optimal number of pending flushes is 0 (or at most a very small number).</td>
</tr>
<tr>
<td>Metric</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Migrations Pending</td>
<td>The number of pending tasks from system methods that have modified the schema. Schema updates have to be propagated to all nodes, so pending tasks for this metric can manifest in schema disagreement errors.</td>
</tr>
<tr>
<td>Misc. Tasks Pending</td>
<td>The number of pending tasks from other miscellaneous operations that are not ran frequently.</td>
</tr>
<tr>
<td>Read Requests Pending</td>
<td>The number of read requests that have arrived into the cluster but are waiting to be handled. During low or moderate read load, you should see 0 pending read operations (or at most a very low number).</td>
</tr>
<tr>
<td>Read Repair Tasks Pending</td>
<td>The number of read repair operations that are queued and waiting for system resources in order to run. The optimal number of pending read repairs is 0 (or at most a very small number). A value greater than 0 indicates that read repair operations are in I/O contention with other operations.</td>
</tr>
<tr>
<td>Replicate on Write Tasks Pending</td>
<td>When an insert or update to a row is written, the affected row is replicated to all other nodes that manage a replica for that row. This is called the ReplicateOnWriteStage. This metric tracks the pending tasks related to this stage of the write process. During low or moderate write load, you should see 0 pending replicate on write tasks (or at most a very low number).</td>
</tr>
<tr>
<td>Request Responses Pending</td>
<td>Streaming of data between nodes happens during operations such as bootstrap and decommission when one node sends large numbers of rows to another node. The metric tracks the progress of the streamed rows from the receiving node.</td>
</tr>
<tr>
<td>Streams Pending</td>
<td>Streaming of data between nodes happens during operations such as bootstrap and decommission when one node sends large numbers of rows to another node. The metric tracks the progress of the streamed rows from the sending node.</td>
</tr>
<tr>
<td>Write Requests Pending</td>
<td>The number of write requests that have arrived into the cluster but are waiting to be handled. During low or moderate write load, you should see 0 pending write operations (or at most a very low number).</td>
</tr>
</tbody>
</table>

**Advanced column family alert metrics**

OpsCenter provides the capability to configure alerts for the following column family metrics. Column family metrics provide a granular level of detail for certain Cassandra metrics as they relate to a particular column family.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Writes</td>
<td>The write load on a column family measured in operations per second. This metric includes all writes to a given column family, including write requests forwarded from other nodes.</td>
</tr>
<tr>
<td>Local Write Latency</td>
<td>The response time in microseconds for successful write operations on a column family. The time period starts when nodes receive a write request, and ends when nodes respond.</td>
</tr>
<tr>
<td>Local Reads</td>
<td>The read load on a column family measured in operations per second. This metric includes all reads to a given column family, including read requests forwarded from other nodes.</td>
</tr>
<tr>
<td>Local Read Latency</td>
<td>The response time in microseconds for successful read operations on a column family. The time period starts when a node receives a read request, and ends when the node responds.</td>
</tr>
<tr>
<td>CF: KeyCache Hits</td>
<td>The number of read requests that resulted in the requested row key being found in the key cache.</td>
</tr>
<tr>
<td>CF: KeyCache Requests</td>
<td>The total number of read requests on the row key cache.</td>
</tr>
</tbody>
</table>
CF: KeyCache Hit Rate
The key cache hit rate indicates the effectiveness of the key cache for a given column family by giving the percentage of cache requests that resulted in a cache hit.

CF: RowCache Hits
The number of read requests that resulted in the read being satisfied from the row cache.

CF: RowCache Requests
The total number of read requests on the row cache.

CF: RowCache Hit Rate
The key cache hit rate indicates the effectiveness of the row cache for a given column family by giving the percentage of cache requests that resulted in a cache hit.

Live Disk Used
The current size of live SSTables for a column family. It is expected that SSTable size will grow over time with your write load, as compaction processes continue doubling the size of SSTables. Using this metric together with SSTable count, you can monitor the current state of compaction for a given column family.

Total Disk Used
The current size of the data directories for the column family including space not reclaimed by obsolete objects.

SSTable Count
The current number of SSTables for a column family. When column family memtables are persisted to disk as SSTables, this metric increases to the configured maximum before the compaction cycle is repeated. Using this metric together with live disk used, you can monitor the current state of compaction for a given column family.

Pending Reads/Writes
The number of pending reads and writes on a column family. Pending operations are an indication that Cassandra is not keeping up with the workload. A value of zero indicates healthy throughput.

CF: Bloom Filter Space Used
The size of the bloom filter files on disk.

CF: Bloom Filter False Positives
The number of false positives, which occur when the bloom filter said the row existed, but it actually did not exist in absolute numbers.

CF: Bloom Filter False Positive Ratio
The fraction of all bloom filter checks resulting in a false positive.

**Advanced system alert metrics**

OpsCenter provides the capability to configure alerts for the following operating system metrics:

- **Linux Metrics**
- **Windows Metrics**
- **Mac OSX Metrics**

As with any database system, Cassandra performance greatly depends on underlying systems on which it is running. To configure advanced system metric alerts, you should first have an understanding of the baseline performance of your hardware and the averages of these system metrics when the system is handling a typical workload.

**Linux metrics**

On Linux, you can configure alerts on memory, cpu and disk events.

**Memory metrics on Linux**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Free</td>
<td>System memory that is not being used.</td>
</tr>
<tr>
<td>Memory Used</td>
<td>System memory used by application processes.</td>
</tr>
</tbody>
</table>
### Memory

<table>
<thead>
<tr>
<th>Memory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffered</td>
<td>System memory used for caching file system metadata and tracking in-flight pages.</td>
</tr>
<tr>
<td>Shared</td>
<td>System memory that is accessible to CPUs.</td>
</tr>
<tr>
<td>Cached</td>
<td>System memory used by the OS disk cache.</td>
</tr>
</tbody>
</table>

### CPU metrics on Linux

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Percentage of time the CPU is idle.</td>
</tr>
<tr>
<td>Iowait</td>
<td>Percentage of time the CPU is idle and there is a pending disk I/O request.</td>
</tr>
<tr>
<td>Nice</td>
<td>Percentage of time spent processing prioritized tasks. Niced tasks are also counted in system and user time.</td>
</tr>
<tr>
<td>Steal</td>
<td>Percentage of time a virtual CPU waits for a real CPU while the hypervisor services another virtual processor.</td>
</tr>
<tr>
<td>System</td>
<td>Percentage of time allocated to system processes.</td>
</tr>
<tr>
<td>User</td>
<td>Percentage of time allocated to user processes.</td>
</tr>
</tbody>
</table>

### Disk metrics on Linux

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Usage</td>
<td>Percentage of disk space Cassandra uses at a given time.</td>
</tr>
<tr>
<td>Free Disk Space</td>
<td>Available disk space in GB.</td>
</tr>
<tr>
<td>Used Disk Space</td>
<td>Used disk space in GB.</td>
</tr>
<tr>
<td>Disk Read Throughput</td>
<td>Average disk throughput for read operations in megabytes per second.</td>
</tr>
<tr>
<td></td>
<td>Exceptionally high disk throughput values may indicate I/O contention.</td>
</tr>
<tr>
<td>Disk Write Throughput</td>
<td>Average disk throughput for write operations in megabytes per second.</td>
</tr>
<tr>
<td>Disk Read Rate</td>
<td>Averaged disk speed for read operations.</td>
</tr>
<tr>
<td>Disk Write Rate</td>
<td>Averaged disk speed for write operations.</td>
</tr>
<tr>
<td>Disk Latency</td>
<td>Average time consumed by disk seeks in milliseconds.</td>
</tr>
<tr>
<td>Disk Request Size</td>
<td>Average size in sectors of requests issued to the disk.</td>
</tr>
<tr>
<td>Disk Queue Size</td>
<td>Average number of requests queued due to disk latency.</td>
</tr>
<tr>
<td>Disk Utilization</td>
<td>Percentage of CPU time consumed by disk I/O.</td>
</tr>
</tbody>
</table>

### Windows metrics

On Windows, you can configure alerts on memory, cpu and disk events.

### Memory metrics on Windows

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Memory</td>
<td>Physical memory that is not being used.</td>
</tr>
<tr>
<td>Pool Nonpaged</td>
<td>Physical memory that stores the kernel and other system data structures.</td>
</tr>
<tr>
<td>Pool Paged Resident</td>
<td>Physical memory allocated to unused objects that can be written to disk to free memory for reuse.</td>
</tr>
<tr>
<td>System Cache Resident</td>
<td>Physical pages of operating system code in the file system cache.</td>
</tr>
</tbody>
</table>

### CPU metrics on Windows
### Disk metrics on Windows

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Percentage of time the CPU is idle.</td>
</tr>
<tr>
<td>Privileged</td>
<td>Percentage of time the CPU spends executing kernel commands.</td>
</tr>
<tr>
<td>User</td>
<td>Percentage of time allocated to user processes.</td>
</tr>
</tbody>
</table>

### Disk metrics on Windows

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Usage</td>
<td>Percentage of disk space Cassandra uses at a given time.</td>
</tr>
<tr>
<td>Free Disk Space</td>
<td>Available disk space in GB.</td>
</tr>
<tr>
<td>Used Disk Space</td>
<td>Used disk space in GB.</td>
</tr>
<tr>
<td>Disk Read Throughput</td>
<td>Average disk throughput for read operations in megabytes per second.</td>
</tr>
<tr>
<td>Disk Write Throughput</td>
<td>Exceptionally high disk throughput values may indicate I/O contention.</td>
</tr>
<tr>
<td>Disk Read Rate</td>
<td>Averaged disk speed for read operations.</td>
</tr>
<tr>
<td>Disk Write Rate</td>
<td>Averaged disk speed for write operations.</td>
</tr>
<tr>
<td>Disk Latency</td>
<td>Average time consumed by disk seeks in milliseconds.</td>
</tr>
<tr>
<td>Disk Request Size</td>
<td>Average size of requests in KB issued to the disk.</td>
</tr>
<tr>
<td>Disk Queue Size</td>
<td>Average number of requests queued due to disk latency.</td>
</tr>
<tr>
<td>Disk Utilization</td>
<td>Percentage of CPU time consumed by disk I/O.</td>
</tr>
</tbody>
</table>

### Mac OSX metrics

On Mac OSX, you can configure alerts on memory, cpu and disk events.

### Memory metrics on Mac OSX

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free Memory</td>
<td>System memory that is not being used.</td>
</tr>
<tr>
<td>Used Memory</td>
<td>System memory that is being used by application processes.</td>
</tr>
</tbody>
</table>

### CPU metrics on Mac OSX

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Percentage of time the CPU is idle.</td>
</tr>
<tr>
<td>System</td>
<td>Percentage of time allocated to system processes.</td>
</tr>
<tr>
<td>User</td>
<td>Percentage of time allocated to user processes</td>
</tr>
</tbody>
</table>

### Disk metrics on Mac OSX

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Usage</td>
<td>Percentage of disk space Cassandra uses at a given time.</td>
</tr>
<tr>
<td>Free Space</td>
<td>Available disk space in GB.</td>
</tr>
<tr>
<td>Used Disk Space</td>
<td>Used disk space in GB.</td>
</tr>
<tr>
<td>Disk Throughput</td>
<td>Average disk throughput for read/write operations in megabytes per second.</td>
</tr>
</tbody>
</table>
Managing backups

Using OpsCenter Enterprise Edition, you can take, schedule, and manage backups across all registered clusters.

A backup is a snapshot of all on-disk data files (SSTable files) stored in the data directory. Backups are taken per keyspace and while the system is online. A backup first flushes all in-memory writes to disk, then makes a hard link of the SSTable files for each keyspace. Backups are stored in the snapshots directory of the column family that’s being snapshotted. For example, /var/lib/cassandra/data/cfs/snapshots.

You must have enough free disk space on the node to accommodate making snapshots of your data files. A single snapshot requires little disk space. However, snapshots will cause your disk usage to grow more quickly over time because a snapshot prevents old obsolete data files from being deleted. OpsCenter Data Backups allows you to specify a schedule to remove old backups and prevent backups from being taken when disk space falls below a specified level.

**Note**

OpsCenter Data Backups does not show or manage manual snapshots taken using the nodetool snapshot command.

Scheduling a backup

To schedule a backup:

1. In the OpsCenter Dashboard, click **Data Backups**.
2. Click **Schedule Backup**.
3. In **Add Backup**, select the backup parameters:
   - **Select a Keyspace to backup** - Select the keyspace that you want to back up.
   - **Schedule** - Select a frequency and timezone for your backup. GMT is the default timezone.
   - **Cleanup** - Choose a frequency to remove old backups. (If not specified, you should manually cleanup snapshots.)
4. Click **Save**.
5. To set the percentage of free disk space at which backups are prevented, click **Configure** and then enter the appropriate information.

   The percentage of free disk space that you set applies to all nodes in the cluster.

Detailed information about the backup is recorded in the **Event Log**.

Data modeling in OpsCenter

In the Data Modeling area of OpsCenter, you can:

- View keyspace properties
- **Create keyspaces**
- **Delete keyspaces**
- **Create new column families**
- **View column family properties**
- **Manage performance metrics** on column families, such as setting, truncating, or deleting metrics

This release of OpsCenter does not support column families that you create using CQL 3. When you select a keyspace in Cassandra that contains a CQL 3 column family, the column family does not appear in OpsCenter.

Keyspace operations
Selecting the Data Modeling area in the OpsCenter console lists the keyspaces in the cluster that you are monitoring. You can create a new keyspace or manage keyspaces.

When you create a keyspace, you give it a name, choose a replica placement strategy, the total number of replicas you want, and how those replicas are divided across your data centers (if you have a multiple data center cluster).

**Creating a keyspace**

To create a new keyspace:

1. Select **Data Modeling** in the OpsCenter console.
2. From the list of keyspaces, select **Add Keyspace** in the Data Modeling section of OpsCenter.
3. Give the keyspace a name. Keyspace names should not contain spaces or special characters or exceed the filename size limit of your operating system (for example, 255 bytes on most Linux file systems). Keyspace names are case sensitive.
4. Set the replica placement strategy. The replica placement strategy (along with the cluster-configured snitch) determines how replicas are placed on nodes throughout the cluster. Use one of three built-in replica placement strategies:
   - **SimpleStrategy** - Single data center, rack unaware replica placement. This is the default strategy.
   - **NetworkTopologyStrategy** - Single or multiple data center, rack aware replica placement. This is the recommended strategy.
   - **OldNetworkTopologyStrategy** - Two data centers only, rack aware replica placement. This strategy is deprecated.
5. Choose how many total copies that you want of your keyspace data (replication factor). The NetworkTopologyStrategy requires you to configure how many replicas you want per data center. The data center name you enter should match the data center name used by your cluster-configured snitch. Make sure to name the data center(s) correctly according to your snitch configuration.
6. If you do not want to start defining column families within your new keyspace right away, uncheck the **I would like to create a Column Family** checkbox.
7. Click **Save Keyspace**.

**Managing keyspaces**

To manage keyspaces:

1. Select **Data Modeling** in the OpsCenter console.
2. From the list of keyspaces, select one of the keyspaces.
   In Settings, the replica placement strategy options for the keyspace appear.
3. From the list of column families (below Settings), select a column family to view its properties and to view or change performance tuning metrics.
4. Click **Add Column Family** to **add a column family** to the keyspace.
5. Click **Delete** to delete the keyspace.

**Column family management**

When you create a column family in Cassandra using an application, the CLI, or CQL 2 or earlier, the column family appears in OpsCenter. You can use Data Modeling to manage the column family.

You can also create one type of column family: the dynamic column family. Dynamic column families are those that do not specify column names or values when the column family is created. An application typically supplies this metadata. CQL 3, the default query language in Cassandra, does not support dynamic column families. Earlier versions of CQL
and the CLI support dynamic column families.

This version of OpsCenter does not support defining static column families (per-column meta data), row key data types, or schema information for super column sub-columns described in Cassandra 1.0, or earlier, documentation on http://www.datastax.com.

Creating a dynamic column family

To create a new dynamic column family:

1. Select Data Modeling in the OpsCenter console.
2. From the list of keyspaces, select the keyspace to contain the column family.
3. Give the column family a name.
   Column family names should not contain spaces or special characters and cannot exceed the filename size limit of your operating system (255 bytes on most Linux file systems).
   By default, column families are created with standard columns (column_type: Standard). If you want a column family containing super columns choose column_type: Super.
4. Use compare_with to set the default data type for column names (or super column names).
   Setting the default data type also sets the column sort order for the column family. For example, choosing LongType would sort the columns within a row in numerical order. The sort order cannot be changed after a column family is created, so choose wisely.
5. Use default_validation_class to set the default data type for column values (or super column sub-column values). Always set this for dynamic column families.
6. Click Save Column Family.

Managing column families

To manage column families:

1. Select Data Modeling in the OpsCenter console.
2. From the list of keyspaces, select a keyspace.
   The #CFs columns shows how many column families each keyspace contains.
3. From the list of the column families, select a column family. Click one of the following buttons:
   - View Metrics
     Presents metrics for a column family. In the Metric Options dialog, select a column family (CF) metric to view. To aggregate measurements across the entire cluster, all nodes in the data center, or in a particular node, select Cluster Wide, All Nodes, or the IP address of a node. At this point, you can add a graph of the measurements to the Performance Metrics area, or choose a different column family to measure.
   - Truncate <column family>
     Deletes all data from the column family but does not delete the column family itself. Removal of the data is irreversible.
   - Delete <column family>
     Completely removes the column family from the keyspace.

Browsing the database

Use the Data Explorer area to browse the keyspaces, column families, and data stored in the Cassandra database of a cluster and to zoom in on the columns of a particular row.

To browse the Cassandra database:
Upgrading Opscenter

1. Select **Data Explorer** in the OpsCenter console.
   A list of keyspaces in the cluster appears. By default, the list includes the OpsCenter keyspace, which contains column families of data in the cluster.

2. Click one of the keyspaces. For example, click the OpsCenter keyspace.
   The list of column families appears.

3. Click a column family. For example, click `events_timeline` and expand the OpsCenter console window, so you can see more values.
   A row keys, columns, and data values of the `events_timeline` column family appear in tabular format. You may notice that some data values are hex-encoded, not human-readable values, and other values are perfectly readable. Internally, Cassandra stores row key names, column names and column values as hex byte arrays. If possible, OpsCenter converts data to text you can read.

4. Click the heading row of the table to see the sort control (the arrow on the right). Toggle the sort control to sort the rows in ascending or descending order.

5. To browse through the data, use the scroll bar or click Next to page through the data.

**To zoom in on a particular row:**

1. Assuming the `events_timeline` column family is still selected, type the key for a row in the **Key** text entry box. For example, type the row key `timeline_start`.
   If you set up a secondary index on the column family, the Key drop-down lists key choices.

2. Click the search control (the hourglass icon).
   The columns and values of the row appear.

**Upgrading Opscenter**

This section contains information about upgrading OpsCenter and OpsCenter agents. The following upgrade paths are available:

- **Upgrading OpsCenter to 2.1.3**
- **Upgrading OpsCenter agents**

Before upgrading, check that the Cassandra or DataStax Enterprise version is compatible with the OpsCenter version you intend to run:

<table>
<thead>
<tr>
<th>OpsCenter Version</th>
<th>Cassandra Version</th>
<th>DataStax Enterprise Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.3</td>
<td>0.8, 1.0, 1.1, 1.2</td>
<td>1.0, 2.0, 2.1, 2.2</td>
</tr>
<tr>
<td>2.1</td>
<td>0.8, 1.0, 1.1</td>
<td>1.0, 2.0, 2.1, 2.2</td>
</tr>
<tr>
<td>2.0</td>
<td>0.8, 1.0, 1.1</td>
<td>1.0, 2.0</td>
</tr>
<tr>
<td>1.4.x</td>
<td>0.7, 0.8, 1.0, 1.1</td>
<td>1.0, 2.0</td>
</tr>
</tbody>
</table>

**Upgrading OpsCenter to 2.1.3**

Use the information in this section to upgrade to OpsCenter 2.1.3 from 2.1, 2.0, 1.4.x, or 1.3.x.

**Note**

As of OpsCenter 1.3, the free OpsCenter Community Edition no longer requires a username and password. Only the OpsCenter Enterprise Edition requires login credentials.

**To upgrade to OpsCenter 2.1.3:**
These steps describe installing the new OpsCenter package and restarting the `opscenterd` daemon.

1. On the OpsCenter daemon host, run the appropriate command to update the packages:

   **Debian/Ubuntu:**
   ```
   # apt-get update
   ```

   **RHEL/CentOS:**
   ```
   # yum clean all
   ```

2. Install the upgraded OpsCenter package:

   **Debian/Ubuntu:**
   ```
   # apt-get install opscenter
   ```

   **RHEL/CentOS:**
   ```
   # yum install opscenter
   ```

3. If the package manager prompts you for options regarding `opscenterd.conf`, choose to keep your currently installed version.

4. Restart the OpsCenter daemon.
   ```
   # service opscenterd restart
   ```

**Upgrading OpsCenter agents**

If OpsCenter agents require upgrading for the new release, you are prompted to do by a **Fix** link located near the top of the OpsCenter console:

![0 of 10 agents connected Fix](image)

For information about installing the upgraded agents, see *Installing OpsCenter agents*.

**OpsCenter reference**

This section contains miscellaneous information about OpsCenter.

- OpsCenter and OpsCenter agent ports
- Starting and stopping OpsCenter and its agents
- Debian and Ubuntu Package install locations
- CentOS, OEL, and RHEL Package install locations
- Binary Tarball distribution install locations

**OpsCenter and OpsCenter agent ports**

The following table lists the default port numbers used by OpsCenter and the OpsCenter Agents:
<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8888</td>
<td>OpsCenter website. The opscenterd daemon listens on this port for HTTP requests coming directly from the browser. Configurable in <code>opscenterd.conf</code>.</td>
</tr>
<tr>
<td>50031</td>
<td>OpsCenter HTTP proxy for Job Tracker. The opscenterd daemon listens on this port for incoming HTTP requests from the browser when viewing the Hadoop Job Tracker page directly. (DataStax Enterprise only)</td>
</tr>
<tr>
<td>61620</td>
<td>OpsCenter monitoring port. The opscenterd daemon listens on this port for TCP traffic coming from the agent.</td>
</tr>
<tr>
<td></td>
<td><strong>OpsCenter agents ports (on the monitored nodes)</strong></td>
</tr>
<tr>
<td>7199</td>
<td>JMX monitoring port. Each agent opens a JMX connection to its local node (the Cassandra or DataStax Enterprise process listening on this port). The JMX protocol requires that the client then reconnect on a randomly chosen port (1024+) after the initial handshake.</td>
</tr>
<tr>
<td>8012</td>
<td>Hadoop Job Tracker client port. The Job Tracker listens on this port for job submissions and communications from task trackers; allows traffic from each Analytics node in a DataStax Enterprise cluster.</td>
</tr>
<tr>
<td>9290</td>
<td>Hadoop Job Tracker Thrift port. The Job Tracker listens on this port for Thrift requests coming from the opscenterd daemon. (DataStax Enterprise only)</td>
</tr>
<tr>
<td>50030</td>
<td>Hadoop Job Tracker website port. The Job Tracker listens on this port for HTTP requests. If initiated from the OpsCenter UI, these requests are proxied through the opscenterd daemon; otherwise, they come directly from the browser. (DataStax Enterprise only)</td>
</tr>
<tr>
<td>50060</td>
<td>Hadoop Task Tracker website port. Each Task Tracker listens on this port for HTTP requests coming directly from the browser and not proxied by the opscenterd daemon. (DataStax Enterprise only)</td>
</tr>
<tr>
<td>61621</td>
<td>OpsCenter agent port. The agents listen on this port for SSL traffic initiated by OpsCenter.</td>
</tr>
<tr>
<td>22</td>
<td>SSH port. Configurable in <code>opscenterd.conf</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Solr Port and Demo applications port</strong></td>
</tr>
<tr>
<td>8983</td>
<td>Solr Port and Demo applications port.</td>
</tr>
<tr>
<td></td>
<td><strong>Cassandra client port</strong></td>
</tr>
<tr>
<td>9160</td>
<td>Each agent makes Thrift requests to its local node on this port. Additionally, the port can be used by the opscenterd daemon to make Thrift requests to each node in the cluster.</td>
</tr>
</tbody>
</table>

**Debian and Ubuntu Package install locations**

These packages are installed into the following directories:

- `/var/lib/opscenter` (SSL certificates for encrypted agent/dashboard communications)
- `/var/log/opscenter` (log directory)
- `/var/run/opscenter` (runtime files)
- `/usr/share/opscenter` (JAR, agent, web application, and binary files)
- `/etc/opscenter` (configuration files)
- `/etc/init.d` (service startup script)

**CentOS, OEL, and RHEL Package install locations**

These packages are installed into the following directories:
Binary Tarball distribution install locations

The tar installation creates the following directories in the `<install_location>` directory:

- `/agent` (agent installation files)
- `/bin` (startup and configuration binaries)
- `/conf` (configuration files)
- `/content` (web application files)
- `/doc` (license files)
- `/lib` and `/src` (library files)
- `/log` (OpsCenter log files)
- `/ssl` (SSL files for OpsCenter to agent communications)

OpsCenter API

The OpsCenter API facilitates the development of websites and programs to retrieve data and perform Cassandra administrative actions. The OpsCenter API includes RESTful requests for programmatically performing the same set of operations as the OpsCenter GUI.

The documentation includes examples of each API method and is organized as follows:

- **Managing Clusters** covers methods for getting information about clusters, such as seed hosts and the jmx port, adding a new cluster to OpsCenter, updating a cluster, and removing a cluster.
- **Retrieving Cluster and Node Information** describes methods for getting more information about clusters, such as the endpoint snitch and partitioner name, information about a particular cluster property, nodes in the cluster, and node properties.
- **Performing Cluster Operations** describes methods for performing garbage compaction, flushing memtables, and other administrative actions on nodes and keyspaces.
- **Managing Keyspaces and Column Families** describes keyspace and column family operations, such as creating new keyspaces and column families.
- **Retrieving Metric Data** explains how to measure the performance of clusters, nodes, and column families.
- **Managing Events and Alerts** includes documentation about methods for getting information about events and alerts and setting alert rules.
- **Managing Schedules and Backups** describes methods for scheduling and running a backup and getting information about existing snapshots.

This document contains the following sections:

**Managing Clusters**

You can manage clusters using these HTTP methods:
### Cluster Management Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List configuration information for all clusters.</td>
<td>GET /cluster-configs</td>
<td></td>
</tr>
<tr>
<td>List configuration information for a cluster.</td>
<td>GET /cluster-configs/{cluster_id}</td>
<td></td>
</tr>
<tr>
<td>Add a new cluster to OpsCenter.</td>
<td>POST /cluster-configs</td>
<td></td>
</tr>
<tr>
<td>Update a cluster configuration.</td>
<td>PUT /cluster-configs/{cluster_id}</td>
<td></td>
</tr>
<tr>
<td>Remove a cluster.</td>
<td>DELETE /cluster-configs/{cluster_id}</td>
<td></td>
</tr>
</tbody>
</table>

#### HTTP Methods

**GET/cluster-configs**

Retrieve a list of all configured clusters.

Returns a dictionary of cluster id (key), ClusterConfig objects (value). Each cluster configuration includes a seed_hosts entry (required).

**ClusterConfig**

```json
{
    <section-name>: {
        <prop-name>: <prop-value>,
        ...
    },
    ...
}
```

**Example:**

```
curl
http://127.0.0.1:8888/cluster-configs
-X GET
```

**Output:**

```json
{
    "Test_Cluster": {
        "cassandra": {
            "agents": {
                "use_ssl": "false"
            },
            "seed_hosts": "localhost"
        },
        "cassandra_metrics": {},
        "jmx": {
            "port": 7199
        }
    },
    "Test_Cluster2": {
        "cassandra": {
            "seed_hosts": "2.3.4.5, 2.3.4.6",
            "api_port": 9160
        },
        "cassandra_metrics": {},
        "jmx": {
            "port": 7199,
            "username": "jmx_user"
        }
    }
}
```
"password": "jmx_pass"
}
]
}

GET/cluster-configs/cluster_id
Retrieve the configuration for a single cluster.

Path: cluster_id -- The key, which identifies a single cluster to fetch, in the dictionary returned by arguments: GET /cluster-configs.

Returns a ClusterConfig.

Example:

```
curl http://127.0.0.1:8888/cluster-configs/Test_Cluster
   -X GET
```

Output:

```
{
   "agents": {},
   "cassandra": {
      "seed_hosts": "1.2.3.4, 1.2.3.5"
   },
   "cassandra_metrics": {},
   "jmx": {
      "port": 7199
   }
}
```

POST/cluster-configs
Add a new cluster for OpsCenter monitoring and administration.

Body: A configuration in the format of ClusterConfig. A seed hosts entry is required.

Responses: 201 -- Cluster was added successfully

Returns the ID of the new cluster.

Example:

```
curl http://127.0.0.1:8888/cluster-configs
   -X POST
   -d "
      "cassandra": {
         "seed_hosts": "localhost"
      },
      "cassandra_metrics": {},
      "jmx": {
         "port": "7199"
      }
   "
```

Output:

"Test_Cluster"
PUT/cluster-configs/cluster_id
Update a cluster configuration.

**Path**  `cluster_id` -- The key, which identifies the cluster to be updated, in the dictionary returned by
**arguments:** GET /cluster-configs.
**Body:** A configuration in the same format as `ClusterConfig`. A seed hosts entry is required.
**Responses:** 204 -- Cluster updated successfully

Example:
```bash
curl http://127.0.0.1:8888/cluster-configs/Test_Cluster
-X PUT
-d '{
   "cassandra": {
      "seed_hosts": "localhost"
   },
   "jmx": {
      "port": "7199"
   }
}'
```

DELETE/cluster-configs/cluster_id
Remove a cluster.

**Path**  `cluster_id` -- The key, which identifies the cluster to be removed, in the dictionary returned by
**arguments:** GET /cluster-configs.
**Responses:** 204 -- Cluster removed successfully

Example:
```bash
curl http://127.0.0.1:8888/cluster-configs/Test_Cluster
-X DELETE
```

**Retrieving Cluster and Node Information**

These resources give you information about your Cassandra or DSE cluster and its nodes. For example, you can get a
map of node information including the current OS load, the data held in storage, or the status of streaming or
compaction operations underway.

<table>
<thead>
<tr>
<th>Cluster and Node Retrieval Methods</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve cluster configuration information.</td>
<td>GET /{cluster_id}/cluster</td>
</tr>
<tr>
<td>Retrieve information about a cluster property.</td>
<td>GET /{cluster_id}/cluster/{property}</td>
</tr>
<tr>
<td>Retrieve information about nodes in the cluster.</td>
<td>GET /{cluster_id}/nodes</td>
</tr>
<tr>
<td>Retrieve information about a specific node.</td>
<td>GET /{cluster_id}/nodes/{node_ip}</td>
</tr>
<tr>
<td>Retrieve information about the property of a node.</td>
<td>GET /{cluster_id}/nodes/{node_ip}/{property}</td>
</tr>
<tr>
<td>Retrieve cluster-wide storage capacity information.</td>
<td>GET /{cluster_id}/storage-capacity</td>
</tr>
</tbody>
</table>

**HTTP Methods**

GET/cluster_id/cluster
Retrieve information about the cluster.
Path  cluster_id -- A key, which identifies the node’s cluster, in the dictionary returned by
arguments:  GET /cluster-configs.

Returns a Cluster object.

Cluster

```
{  
   "endpoint_snitch": Full class name of snitch,  
   "name": Name of the cluster,  
   "partitioner": Full class name of partitioner
}
```

Example:

```
curl  
    http://127.0.0.1:8888/Test_Cluster/cluster  
    -X GET
```

Output:

```
{
   "endpoint_snitch": "org.apache.cassandra.locator.SimpleSnitch",  
   "name": "Test Cluster",  
   "partitioner": "org.apache.cassandra.dht.RandomPartitioner"
}
```

GET/cluster_id/cluster/property

Retrieve a single cluster property.

Path  cluster_id -- A key, which identifies the node’s cluster, in the dictionary returned by
arguments:  GET /cluster-configs.

property -- A property of Cluster.

Returns a string.

Example:

```
curl  
    http://127.0.0.1:8888/Test_Cluster/cluster/partitioner  
    -X GET
```

Output:

"org.apache.cassandra.dht.RandomPartitioner"

GET/cluster_id/nodes

Retrieve a list of all nodes and node properties in the cluster.

Path  cluster_id -- A key, which identifies the node’s cluster, in the dictionary returned by
arguments:  GET /cluster-configs.

Returns a list of Node objects.

Node

A property is null if it is not applicable to the node or if there is no data to report.

```
{
   "node_ip": <value>,  
   "node_name": <value>,  
   "token": <value>,
```

55
This table describes the property values of a **Node**:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_ip</td>
<td>IP address</td>
</tr>
<tr>
<td>node_name</td>
<td>Hostname</td>
</tr>
<tr>
<td>token</td>
<td>Token assignment (string)</td>
</tr>
<tr>
<td>node_version</td>
<td>Version of Cassandra (a string) or DataStax Enterprise (dictionary of {&lt;component&gt;:version string})</td>
</tr>
<tr>
<td>load</td>
<td>OS load, which corresponds to the command, 1min avg from uptime</td>
</tr>
<tr>
<td>data_held</td>
<td>Amount of Cassandra data on the node (bytes),</td>
</tr>
<tr>
<td>mode</td>
<td>Examples are &quot;normal&quot;, &quot;decommissioned&quot;, &quot;leaving&quot;. Controlled by Cassandra.</td>
</tr>
<tr>
<td>streaming</td>
<td>Dictionary of active streams - {&lt;destination_node_ip&gt;: percentage progress 0.0-1.0, ...} Example: 1.0 = 100%, .8 = 80%, etc.</td>
</tr>
<tr>
<td>task_progress</td>
<td>Dictionary of active tasks - {&lt;task-name-string&gt;: percentage progress, ...} For example, &quot;major-compaction&quot;, &quot;minor-compaction&quot; is a task name. Percentage of progress is the same as it is for streaming.</td>
</tr>
<tr>
<td>last_seen</td>
<td>Null if the node is up or a UNIX timestamp indicating when it went down</td>
</tr>
<tr>
<td>num_procs</td>
<td>Number of processors</td>
</tr>
<tr>
<td>rpc_ip</td>
<td>The rpc_address set in cassandra.yaml, which is usually the same as the node_ip property</td>
</tr>
<tr>
<td>dc</td>
<td>Name of node's data center</td>
</tr>
<tr>
<td>rack</td>
<td>Name of node's rack</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>network_interfaces</td>
<td>Array of network interface names</td>
</tr>
<tr>
<td>partitions</td>
<td>data</td>
</tr>
<tr>
<td>commitlog</td>
<td>Partition where the commitlog resides</td>
</tr>
<tr>
<td>saved_caches</td>
<td>Partition where caches are saved</td>
</tr>
<tr>
<td>other</td>
<td>List of other partitions used by Cassandra</td>
</tr>
<tr>
<td>devices</td>
<td>data</td>
</tr>
<tr>
<td>commitlog</td>
<td>Device where the commitlog resides</td>
</tr>
<tr>
<td>saved_caches</td>
<td>Device where caches are saved</td>
</tr>
<tr>
<td>other</td>
<td>List of other devices used by Cassandra</td>
</tr>
<tr>
<td>os</td>
<td>Name of the operating system that will contain &quot;linux&quot;, &quot;mac&quot;, or &quot;windows&quot;</td>
</tr>
</tbody>
</table>

**Example:**

```bash
curl http://127.0.0.1:8888/Test_Cluster/nodes -X GET
```

**Output:**

```json
{
  "data_held": 53067368.0,
  "dc": "Cassandra",
  "devices": {
    "commitlog": "disk0",
    "data": [
      "disk0"
    ],
    "other": [],
    "saved_caches": "disk0"
  },
  "last_seen": 0,
  "load": 0.20000000000000001,
  "mode": "normal",
  "network_interfaces": [
    "lo0",
    "en0",
    "en1"
  ],
  "node_ip": "1.2.3.4",
  "node_name": "John-Does-MacBook-Pro.local",
  "node_version": {
    "cassandra": "1.0.8",
    "dse": "2.0-1",
    "jobtracker": null,
    "search": null,
    "tasktracker": null
  },
  "num_procs": 8,
  "os": "mac os x",
  "partitions": {
    "commitlog": "/dev/disk0s2",
```
GET/cluster_id/nodes/node_ip

Retrieve data for a specific node.

Path arguments:

- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- **node_ip** -- IP address -- the value of node_ip in the Node.

Returns a Node object.

Example:

curl http://127.0.0.1:8888/Test_Cluster/nodes/192.168.1.28
-X GET

Output:

```json
{
  "data_held": 52166921.0,
  "dc": "Cassandra",
  "devices": {
    "commitlog": "disk0",
    "data": [
      "disk0"
    ],
    "other": [],
    "saved_caches": "disk0"
  },
  "last_seen": 0,
  "load": 0.080000000000000002,
  "mode": "normal",
  "network_interfaces": [
    "lo0",
    "en0",
    "en1"
  ],
  "node_ip": "1.2.3.4",
  "node_name": "John-Does-MacBook-Pro.local",
  "node_version": {
    "cassandra": "1.0.8",
    "dse": "2.0-1",
  }
}
```
GET/cluster_id/nodes/node_ip/property
Retrieve a single property for a node.

Path arguments:
- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- **node_ip** -- IP address -- the value of node_ip in the Node.
- **property** -- Node property name -- the value of a property, such as token, in the Node.

Returns a single property from a Node object.

Example:

curl http://127.0.0.1:8888/Test_Cluster/nodes/1.2.3.4/token
-X GET

Output:
"34478773810192488084662817292306645152"

GET/cluster_id/storage-capacity
Retrieve cluster-wide storage capacity information.

Path arguments:
- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.

Returns:

```
{
  "free_gb": Total free space in the cluster in GB,
  "used_gb": Total used space in the cluster in GB,
  "reporting_nodes": How many nodes are included in the first 2 numbers
}
```

Example:
curl
  http://127.0.0.1:8888/Test_Cluster/storage-capacity
-X GET

Output:
{
  "free_gb": 627,
  "reporting_nodes": 1,
  "used_gb": 70
}

**Performing Cluster Operations**

Cluster operations include initiating administrative actions on nodes, such as garbage compaction, in a Cassandra or DSE cluster, rebalancing a cluster, and managing API requests sent to cluster.

<table>
<thead>
<tr>
<th>Node Administration Methods</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiate Java garbage compaction on a node.</td>
<td>GET /{cluster_id}/ops/gc/{node_ip}</td>
</tr>
<tr>
<td>Assign a new token to the node.</td>
<td>PUT /{cluster_id}/ops/move/{node_ip}</td>
</tr>
<tr>
<td>Flush all memtables from the node.</td>
<td>GET /{cluster_id}/ops/drain/{node_ip}</td>
</tr>
<tr>
<td>Decommission a node.</td>
<td>GET /{cluster_id}/ops/decommission/{node_ip}</td>
</tr>
<tr>
<td>Clean up a keyspace.</td>
<td>POST /{cluster_id}/ops/cleanup/{node_ip}/{ks_name}</td>
</tr>
<tr>
<td>Flush memtables from a keyspace.</td>
<td>POST /{cluster_id}/ops/flush/{node_ip}/{ks_name}</td>
</tr>
<tr>
<td>Repair a keyspace.</td>
<td>POST /{cluster_id}/ops/repair/{node_ip}/{ks_name}</td>
</tr>
<tr>
<td>Perform compaction on a keyspace.</td>
<td>POST /{cluster_id}/ops/compact/{node_ip}/{ks_name}</td>
</tr>
</tbody>
</table>

Ensure that each node in a Cassandra cluster is managing an equal amount of data using these methods.

<table>
<thead>
<tr>
<th>Cluster Rebalancing Methods</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>List moves to balance a cluster.</td>
<td>GET /{cluster_id}/ops/rebalance</td>
</tr>
<tr>
<td>Run a list of moves to balance a cluster.</td>
<td>POST /{cluster_id}/ops/rebalance</td>
</tr>
</tbody>
</table>

Determine the status of Real-time Query operations in Cassandra using these methods.

<table>
<thead>
<tr>
<th>Request Management Methods</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the status of the request.</td>
<td>GET /{cluster_id}/request/{request_id}/status</td>
</tr>
<tr>
<td>Cancel a request.</td>
<td>POST /{cluster_id}/request/{request_id}/cancel</td>
</tr>
</tbody>
</table>

**Node Administration Methods**

GET/{cluster_id}/ops/gc/{node_ip}
Initiate Java garbage compaction on a Node.

Path arguments:
- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- **node_ip** -- IP address of the target Node.

Returns null.

Example:
PUT/cluster_id/ops/move/node_ip
Assign a new token to the node.

Path arguments:
- cluster_id -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- node_ip -- Node to be assigned a new token.

Body: New token to assign to node

Returns a Request ID, the ID returned by the API call that triggered the request.

Example:

curl
http://127.0.0.1:8888/Test_Cluster/ops/move/10.11.12.72
-X PUT
-d "85070591730234615865843651857942052864"

Output:
"72ff69b2-9cf5-4777-a600-9173b3fe7e6a"

GET/cluster_id/ops/drain/node_ip
Initiate a drain operation to flush all memtables from the node.

Path arguments:
- cluster_id -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- node_ip -- Node to be flushed of memtables.

Returns null.

Example:

curl
http://127.0.0.1:8888/Test_Cluster/ops/drain/1.2.3.4
-X GET

GET/cluster_id/ops/decommission/node_ip
Initiate decommissioning of a node.

Path arguments:
- cluster_id -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- node_ip -- Node to be decommissioned.

Returns null.

Example:

curl
http://127.0.0.1:8888/Test_Cluster/ops/decommission/1.2.3.4
-X GET

POST/cluster_id/ops/cleanup/node_ip/ks_name
Initiate a cleanup operation for the specified keyspace.
Path arguments:

- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by `GET /cluster-configs`.
- **node_ip** -- Node that initiates cleaning of the keyspace.
- **ks_name** -- Name of the keyspace to be cleaned.

**Body:** List of column families to cleanup. Empty = all column families.

Example

```
curl http://127.0.0.1:8888/Test_Cluster/ops/cleanup/1.2.3.4/Keyspace1
-X POST
-d '"ColFam1", "ColFam2"'
```

**POST/cluster_id/ops/flush/node_ip/ks_name**
Flush memtables for a keyspace.

Path arguments:

- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by `GET /cluster-configs`.
- **node_ip** -- Node to be flushed of memtables for a keyspace.
- **ks_name** -- Keyspace of the memtables to be flushed.

**Body:** List of column families to flush. Empty = all column families.

Example

```
curl http://127.0.0.1:8888/Test_Cluster/ops/flush/1.2.3.4/Keyspace1
-X POST
-d '"ColFam1", "ColFam2"'
```

**POST/cluster_id/ops/repair/node_ip/ks_name**
Initiates repair of a keyspace.

Path arguments:

- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by `GET /cluster-configs`.
- **node_ip** -- Node that initiates repair.
- **ks_name** -- Keyspace to be repaired.

**Body:** List of column families to repair. Empty = all column families.

Example

```
curl http://127.0.0.1:8888/Test_Cluster/ops/repair/1.2.3.4/Keyspace1
-X POST
-d '"ColFam1", "ColFam2"'
```

**POST/cluster_id/ops/compact/node_ip/ks_name**
Initiates a major compaction on a keyspace.
Path arguments:

- **cluster_id** -- A key, which identifies the node’s cluster, in the dictionary returned by GET /cluster-configs.
- **node_ip** -- Node that initiates the compaction.
- **ks_name** -- Keyspace to be compacted.

**Body**: List of column families to compact. Empty = all column families.

Returns null.

**Example**

curl
http://127.0.0.1:8888/Test_Cluster/ops/compact/1.2.3.4/Keyspace1
-X POST
-d '"["ColFam1", "ColFam2"]"'

---

**Cluster Rebalancing Methods**

**GET/cluster_id/ops/rebalance**

Return a list of proposed moves to run to balance a cluster.

Path arguments:

- **cluster_id** -- A key, which identifies the node’s cluster, in the dictionary returned by GET /cluster-configs.

Returns a list of moves, where each move is a token and the IP address of its assigned node. The result of this call is passed to POST /{cluster_id}/ops/rebalance.

**Example**

curl http://127.0.0.1:8888/Test_Cluster/ops/rebalance
-X GET

**Output**:

```
[
  [
    "85070591730234615865843651857942052864",
    "10.11.12.152"
  ]
]
```

**POST/cluster_id/ops/rebalance**

Run the specified list of moves to balance a cluster.

Path arguments:

- **cluster_id** -- A key, which identifies the node’s cluster, in the dictionary returned by GET /cluster-configs.

**Query params**:

- **sleep** -- An optional number of seconds to wait between each move.

**Body**: A list of moves to run to balance this cluster. This is typically the result of GET /{cluster_id}/ops/rebalance.

Returns a Request ID for determining the status of, or cancelling, a running rebalance.

**Example**

curl http://127.0.0.1:8888/Test_Cluster/ops/rebalance
-X POST
-d "{{
  "85070591730234615865843651857942052864",
  "10.11.12.152"
}}"
Request Management Methods

GET/cluster_id/request/request_id/status
Check the status of an asynchronous request sent to OpsCenter.

Path arguments:
- `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- `request_id` -- The ID returned by the API call that triggered the request.

Return a dictionary describing the status of the request.

Example
```
curl http://127.0.0.1:8888/Test_Cluster/request/6b6b15aa-df8a-43f1-aab3-efce6b8589e4/status
-X GET
```

```
{
  "status": "running",
  "started": 1334856122,
  "error_message": null,
  "finished": null,
  "moves": [
    {
      "status": null,
      "ip": "10.100.100.100",
      "old": "2",
      "new": "85070591730234615865843651857942052864"
    }
  ],
  "id": "6b6b15aa-df8a-43f1-aab3-efce6b8589e4"
}
```

POST/cluster_id/request/request_id/cancel
Cancel an asynchronous request sent to OpsCenter.

Path arguments:
- `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- `request_id` -- The ID returned by the API call that triggered the request.

Returns null.

Example
```
curl http://127.0.0.1:8888/Test_Cluster/request/6b6b15aa-df8a-43f1-aab3-efce6b8589e4/cancel
-X POST
```

The request is canceled.

Managing Keyspaces and Column Families

You can perform keyspace and column family operations, such as adding and getting information about keyspaces and column families, using these HTTP methods:
Keyspace and Column Family Management Methods | URL
--- | ---
Retrieve a list of all keyspaces. | GET /{cluster_id}/keyspaces
Retrieve information about a keyspace. | GET /{cluster_id}/keyspaces/{ks_name}
Retrieve a single property about a keyspace | GET /{cluster_id}/keyspaces/{ks_name}/{attribute}
Add a keyspace to the cluster. | POST /{cluster_id}/keyspaces/{ks_name}
Add a column family to a keyspace. | POST /{cluster_id}/keyspaces/{ks_name}/cf/{cf_name}
Get the description of a column family. | GET /{cluster_id}/keyspaces/{ks_name}/cf/{cf_name}
Truncate a column family. | DELETE /{cluster_id}/data/{ks_name}/cf/{cf_name}
Delete a column family itself. | DELETE /{cluster_id}/keyspaces/{ks_name}/cf/{cf_name}
Delete a keyspace in a cluster. | DELETE /{cluster_id}/keyspaces/{ks_name}

### HTTP Methods

**GET/cluster_id/keyspaces**
Retrieve all configured keyspaces in the cluster.

**Path** | **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
--- | ---
**arguments:**
GET /cluster-configs.

**Query params:**
- **ksfields** -- Optional. Comma delimited list of explicit keyspace properties to return.
- **cffields** -- Optional. Comma delimited list of explicit column family properties to return.

Returns a dictionary where the key is the keyspace name, and the value is a dictionary of its properties. The list of properties for keyspaces and column families depends on the version of DataStax Enterprise/Cassandra that you're running.

**Example:**

```
curl  
http://127.0.0.1:8888/Test_Cluster/keyspaces  
-X GET
```

**Output:**

```json
{
   "Keyspace1": {
      "column_families": {
         "ColFam1": {
            "column_type": "Standard",
            "min_compaction_threshold": 4,
            ...
         },
         ...
      },
      "replica_placement_strategy": "org.apache.cassandra.locator.SimpleStrategy",
      "strategy_options": {
         "repl -----
GET/cluster_id/keyspaces/ks_name
Retrieve information about a specific keyspace in the cluster.

Path arguments:
- `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- `ks_name` -- The value of a 'keyspace' property in the output of GET /{cluster_id}/keyspaces.

Query params:
- `ksfields` -- Optional. Comma delimited list of explicit keyspace properties to return.
- `cffield` -- Optional. Comma delimited list of explicit column family properties to return.

Returns the dictionary containing all keyspace properties. The properties returned depend on the version of DataStax Enterprise/Cassandra.

Example

curl
http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspace1
-X GET

Output:
{
  "column_families": {
    "ColFam1": {
      "column_type": "Standard",
      "min_compaction_threshold": 4,
      ...
    },
    ...
  },
  "replica_placement_strategy": "org.apache.cassandra.locator.SimpleStrategy",
  "strategy_options": {
    "replication_factor": "1"
  },
  ...
}

GET/cluster_id/keyspaces/ks_name/attribute
Retrieve a single property of a keyspace.

Path arguments:
- `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- `ks_name` -- The value of a 'keyspace' property in the output of GET /{cluster_id}/keyspaces.
- `attribute` -- One of the keys returned by GET /{cluster_id}/keyspaces/{ks_name}.

Example

curl
http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspacel/replica_placement_strategy
-X GET

Output:
"org.apache.cassandra.locator.SimpleStrategy"
POST/cluster_id/keyspaces/ks_name
Add a keyspace to the cluster.

Path
arguments:
  • cluster_id -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
  • ks_name -- The name of the keyspace to add to the cluster.

Body: A json dictionary describing the attributes of the keyspace you are adding.

Responses: 200 -- Keyspace created successfully

The CQL keyspace storage parameters have keys corresponding to Cassandra keys in the keyspace dictionary and are valid attributes for strategy_class and strategy_options parameters:

<table>
<thead>
<tr>
<th>Keyspace Storage Parameter</th>
<th>Keyspace Dictionary Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategy_class</td>
<td>replica_placement_strategy</td>
</tr>
<tr>
<td>strategy_options</td>
<td>strategy_options</td>
</tr>
</tbody>
</table>

Example

curl
  http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspace2
  -X POST
  -d
  '{"strategy_class": "org.apache.cassandra.locator.SimpleStrategy",
   "strategy_options": {"replication_factor": "1"}}'
Path arguments:

- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- **ks_name** -- The name of the keyspace that contains the column family.
- **cf_name** -- The name of the column family.

Returns a dictionary describing the requested column family. Properties returned depend on the DataStax Enterprise/Cassandra version.

**Example**

curl

```
http://127.0.0.1:8888/Test_Cluster/keyspaces/test_ks/cf/users
-X GET
```

Output:

```
{
  "column_type": "Standard",
  "min_compaction_threshold": 4,
  ...
}
```

DELETE/cluster_id/data/ks_name/cf_name

Truncate an existing column family. Deletes all data from the column family but does not delete the column family itself.

Path arguments:

- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- **ks_name** -- The name of the keyspace containing the column family.
- **cf_name** -- The name of the column family to truncate.

Returns null.

**Example**

curl

```
http://127.0.0.1:8888/Test_Cluster/data/test_ks/users
-X DELETE
```

DELETE/cluster_id/keyspaces/ks_name/cf/cf_name

Delete a column family.

Path arguments:

- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- **ks_name** -- The name of the keyspace containing the column family.
- **cf_name** -- The name of the column family to delete.

Responses:

- 204 -- Column family deleted successfully

**Example**

curl

```
http://127.0.0.1:8888/Test_Cluster/keyspaces/test_ks/cf/users
-X DELETE
```

DELETE/cluster_id/keyspaces/ks_name

Delete an existing keyspace in the cluster.
Path arguments:

- **cluster_id** -- A key, which identifies the node’s cluster, in the dictionary returned by GET /cluster-configs.

- **ks_name** -- The name of the keyspace to delete.

Responses: 204 -- Keyspace deleted successfully

Example

```
curl http://127.0.0.1:8888/Test_Cluster/keyspaces/test_ks
-X DELETE
```

**Retrieving Metric Data**

Using the metric retrieval methods you can retrieve performance metrics at the cluster, node, and column family levels.

<table>
<thead>
<tr>
<th>Metric Retrieval Methods</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve cluster-wide metrics.</td>
<td>GET /{cluster_id}/cluster-metrics/{dc}/{metric}</td>
</tr>
<tr>
<td>Retrieve cluster-wide metrics about a device.</td>
<td>GET /{cluster_id}/cluster-metrics/{dc}/{metric}/{device}</td>
</tr>
<tr>
<td>Retrieve cluster-wide metrics about a column family.</td>
<td>GET /{cluster_id}/cluster-metrics/{dc}/{ks_name}/{cf_name}/{metric}</td>
</tr>
<tr>
<td>Retrieve metrics about a node.</td>
<td>GET /{cluster_id}/metrics/{node_ip}/{metric}</td>
</tr>
<tr>
<td>Retrieve node-specific metrics about a device.</td>
<td>GET /{cluster_id}/metrics/{node_ip}/{metric}/{device}</td>
</tr>
<tr>
<td>Retrieve node-specific metrics about a column family.</td>
<td>GET /{cluster_id}/metrics/{node_ip}/{ks_name}/{cf_name}/{metric}</td>
</tr>
</tbody>
</table>

You can choose from a large number of **metric keys** an option to pass with these methods, making retrieval of a wide spectrum of performance information possible.

**Filtering the Metric Data Output**

You can also use the following query parameters with these methods to filter the output:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>(optional) A timestamp in seconds indicating the beginning of a range for aggregating the metric.</td>
</tr>
<tr>
<td>end</td>
<td>(optional) A timestamp in seconds indicating the end of a range for aggregating the metric.</td>
</tr>
<tr>
<td>step</td>
<td>(optional) The resolution of the input. Valid input options are: 1, 5, 120, or 1440 minutes; corresponding output intervals are 60, 200, 7200, or 86400 seconds.</td>
</tr>
<tr>
<td>function</td>
<td>(optional) The type of aggregation to perform on the metric: min, max, or average. By default, results are returned for all three types of aggregation.</td>
</tr>
</tbody>
</table>

Results of calls to retrieve metrics are returned in the following format:

```
{
  [<node_ip>: | <device>: | <keyspace.columnfamily>:]
    {
      <function>:
        [
          [<timestamp> <value>],
          ...
        ]
    }
}
```

By default, the output is metric data points at 60-second intervals over a 24-hour period.
GET/cluster_id/cluster-metrics/dc/metric

Aggregates a metric across multiple nodes in the cluster rather than retrieving data about a single node.

Path arguments:
- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- **dc** -- The name of the data center for the nodes. Use the name all to aggregate a metric across all data centers.
- **metric** -- One of the Cluster Metrics Keys.

Query params:
- **parameters** -- The parameters listed in Filtering the Metric Data Output.

Returns metric data across multiple nodes in a cluster.

Example
Get the average write requests per second over to the cluster over all data centers on May 1, 2012 from 8 AM to 5 PM GMT. Show data points at 2-hour (120-minute) intervals.

curl http://127.0.0.1:8888/Test_Cluster/cluster-metrics/all/write-ops
-d 'step=120'
-d 'start=1335859200'
-d 'end=1335891600'
-d 'function=average'
-G -X GET

Output:
Data points at 2-hour (7200 seconds) intervals show the number of write requests per second during business hours on May 1.

```
{
  "Total": {
    "AVERAGE": [
      [1335859200, null],
      [1335866400, 13.376885890960693],
      [1335873600, 13.372154712677002],
      [1335880800, 13.365732669830322],
      [1335888000, 13.392115592956543]
    ]
  }
}
```
Aggregate a disk or network metric, which pertains to a specific device, across multiple nodes in the cluster rather than retrieving data about a single node.

Path arguments:
- `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by `GET /cluster-configs`.
- `dc` -- The name of the data center for the nodes. Use the name `all` to aggregate a metric across all data centers.
- `metric` -- One of the [Cluster Metrics Keys](#) or [Operating System Metrics Keys](#).
- `device` -- The device to be measured, which the Node object lists. Use the name `all` to measure all devices, for example all disk devices associated with a disk metric.

Query params:
- `parameters` -- The parameters listed in [Filtering the Metric Data Output](#).

Examples of Device Arguments

<table>
<thead>
<tr>
<th>Example devices</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;network_interfaces&quot;: [&quot;lo0&quot;, &quot;en1&quot;]</td>
<td>Devices measured by network metrics</td>
</tr>
<tr>
<td>&quot;devices&quot;: {&quot;saved_caches&quot;: &quot;disk1&quot;, &quot;commitlog&quot;: &quot;disk1&quot;, &quot;other&quot;: [&quot;disk0&quot;], &quot;data&quot;: [&quot;disk1&quot;]}</td>
<td>Devices measured by disk metrics</td>
</tr>
<tr>
<td>&quot;partitions&quot;: {&quot;saved_caches&quot;: &quot;/dev/disk1s2&quot;, &quot;commitlog&quot;: &quot;/dev/disk1s2&quot;, &quot;other&quot;: [&quot;/dev/disk0s2&quot;], &quot;data&quot;: [&quot;/dev/disk1s2&quot;]}</td>
<td>Devices measured by partition metrics</td>
</tr>
</tbody>
</table>

Using a partition, network interface, or other device name for the device argument returns disk or network metric data about a specific device across multiple nodes. Using `all` for the device name returns a dictionary of keys (device names) and the values (results for that device).

Example

Get the average GB of space on all disks in all data centers used each day by the cluster from April 11, 2012 00:00:00 to April 26, 2012 00:00:00 GMT.

```bash
curl http://127.0.0.1:8888/Test_Cluster/cluster-metrics/all/os-disk-used/all
   -d 'step=1440'
   -d 'start=1334102400'
   -d 'end=1335398400'
   -d 'function=average'
   -G -X GET
```

Output:

```json
{
   "Total": {
      "AVERAGE": [
      [1334102400, null],
      [1334188800, 21.000694274902344],
      [1334275200, 8.736943244934082],
      [1334361600,
```
9.0
],
[ 133448000, 19.0
],
[ 1334534400, 19.0
],
[ 1334620800, 19.0
],
[ 1334707200, 19.0
],
[ 1334793600, 18.629029273986816
],
[ 1334880000, 19.923184394836426
],
[ 1334966400, 25.0
],
[ 1335052800, 25.0
],
[ 1335139200, 25.923053741455078
],
[ 1335225600, 26.0
],
[ 1335312000, 26.549484252929688
]
}

GET/cluster_id/cluster-metrics/dc/ks_name/cf_name/metric
Aggregate a column family metric across multiple nodes in the cluster rather than retrieving data about a single node.
Path arguments:

- **cluster_id** -- A key, which identifies the node’s cluster, in the dictionary returned by GET /cluster-configs.
- **dc** -- The name of the data center for the nodes. Use the name all to aggregate a metric across all data centers.
- **ks_name** -- The keyspace that contains the column family to be measured.
- **cf_name** -- The column family to be measured.
- **metric** -- One of the Column Family Metrics Keys.

Query params:

- **parameters** -- The parameters listed in Filtering the Metric Data Output.

Returns metric data for multiple nodes.

**Example**

Get the maximum bytes of disk space used for live data by the rollups60 column family in the Opscenter keyspace of the cluster over all data centers from May 1, 2012 00:00:00 to May 5, 2012 00:00:00 GMT.

```bash
curl http://127.0.0.1:8888/Test_Cluster/cluster-metrics/all/OpsCenter/rollups60/cf-live-disk-used
-d 'function=max'
-d 'start=1335830400'
-d 'end=1336176000'
-d 'step=1440'
-G -X GET
```

Output:

Data points at 24-hour intervals show the metrics for the period.

```
{

"Total": {
   "MAX": [
      [1335830400, 9740462592.0],
      [1335916800, 9932527616.0],
      [1336003200, null],
      [1336089600, 1064448512.0]
   ]
}
```

**GET/cluster_id/metrics/node_ip/metric**

Retrieve metric data for a single node.
Path arguments:

- **cluster_id** -- A key, which identifies the node’s cluster, in the dictionary returned by GET /cluster-configs.
- **node_ip** -- IP address of the target Node.
- **metric** -- One of the Cluster Metrics Keys.

Query params:

- **parameters** -- The parameters listed in Filtering the Metric Data Output.

Returns metric data for a single node.

**Example**

Get the daily average data load on cluster node 10.11.12.150 from April 20, 2012 00:00:00 to April 26, 2012 00:00:00 GMT.

```
curl http://127.0.0.1:8888/Test_Cluster/metrics/10.11.12.150/data-load
  -d 'step=1440'
  -d 'start=1334880000'
  -d 'end=1335398400'
  -d 'function=average'
  -G -X GET
```

Output:

```
{
  "10.11.12.150": {
    "AVERAGE": [
      [1334880000, null],
      [1334966400, 6353770496.0],
      [1335052800, 6560092672.0],
      [1335139200, 6019291136.0],
      [1335225600, 6149050880.0],
      [1335312000, 6271239680.0]
    ]
  }
}
```

**GET/cluster_id/metrics/node_ip/metric/device**

Aggregate a disk or network metric for a single node.
Path arguments:

- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- **node_ip** -- IP address of the target Node.
- **metric** -- One of the Cluster Metrics Keys or Operating System Metrics Keys.
- **device** -- The device to be measured. Use the name all to measure all devices associated with a disk metric. See GET /{cluster_id}/cluster-metrics/{dc}/{metric}/{device} for examples of devices.

Query params: parameters -- The parameters listed in Filtering the Metric Data Output.

Returns disk or network metrics data for a single node.

**Example**
Get the maximum GB of disk space for all disks used by cluster node 10.11.12.150 from April 30, 2012 at 22:05 to May 1, 2012 8:00:00 GMT.

```bash
curl http://127.0.0.1:8888/Test_Cluster/metrics/10.11.12.150/os-disk-used/all
   -d 'start=1335823500'
   -d 'end=1335859200'
   -d 'step=120'
   -d 'function=max'
-G -X GET
```

Output:
Data points at 2-minute intervals show the disk space used by device /dev/sda1.

```json
{
    "/dev/sda1": {
        "MAX": [
            [1335823200, null],
            [1335830400, 17.0],
            [1335837600, 16.0],
            [1335844800, 17.0],
            [1335852000, 16.0]
        ]
    }
}
```

GET/cluster_id/metrics/node_ip/ks_name/cf_name/metric
Retrieve metric data about a column family on a single node.
Path arguments:
- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- **node_ip** -- IP address of the target Node.
- **ks_name** -- The keyspace that contains the column family to be measured.
- **cf_name** -- The column family to be measured.
- **metric** -- One of the Column Family Metrics Keys.

Query params:
- **parameters** -- The parameters listed in Filtering the Metric Data Output.

Example
Get the daily, maximum response time (in microseconds) to write requests on the rollups60 column family in the OpsCenter keyspace by cluster node 10.11.12.150 from May 1, 2012 at 00:00:00 to May 5, 2012 00:00:00 GMT.

```bash
curl http://127.0.0.1:8888/Test_Cluster/metrics/10.11.12.150/OpsCenter/rollups60/cf-write-latency-op
  -d 'function=max'
  -d 'start=1335830400'
  -d 'end=1336176000'
  -d 'step=1440'
  -G -X GET
```

Output:
```
{
  "OpsCenter.rollups60": {
    "MAX": [
      [1335830400, 102.28681945800781],
      [1335916800, 124.86614227294922],
      [1336003200, null],
      [1336089600, 127.14733123779297]
    ]
  }
}
```

**Metrics Attribute Key Lists**
This section contains these tables of metric keys to use with resources that retrieve OpsCenter performance data:

- Cluster Metrics Keys
- Column Family Metrics Keys
- Operating System Metrics Keys

**Cluster Metrics Keys**
This list of keys corresponds to Cassandra metrics collected by OpsCenter:

<table>
<thead>
<tr>
<th>Key</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data-load</td>
<td>Bytes</td>
<td>Size of the data on the node.</td>
</tr>
<tr>
<td>pending-compaction-tasks</td>
<td>--</td>
<td>Number of compaction operations queued and waiting to run.</td>
</tr>
<tr>
<td>pending-flush-sorter-tasks</td>
<td>--</td>
<td>Number of pending tasks related to the first step in flushing memtables to disk as SSTables.</td>
</tr>
<tr>
<td>pending-flushes</td>
<td>--</td>
<td>Number of memtables queued for the flush process.</td>
</tr>
<tr>
<td>pending-gossip-tasks</td>
<td>--</td>
<td>Number of gossip messages and acknowledgments queued and waiting to be sent or received.</td>
</tr>
<tr>
<td>pending-hinted-handoff</td>
<td>--</td>
<td>Number of hints in the queue waiting to be delivered after a failed node comes up.</td>
</tr>
<tr>
<td>pending-internal-responses</td>
<td>--</td>
<td>Number of pending tasks from internal tasks, such as nodes joining and leaving the cluster.</td>
</tr>
<tr>
<td>pending-memtable-post-flushers</td>
<td>--</td>
<td>Number of pending tasks related to the last step in flushing memtables to disk as SSTables.</td>
</tr>
<tr>
<td>pending-migrations</td>
<td>--</td>
<td>Number of pending tasks from system methods that modified the schema.</td>
</tr>
<tr>
<td>pending-misc-tasks</td>
<td>--</td>
<td>Number of pending tasks from infrequently run operations, not measured by another metric.</td>
</tr>
<tr>
<td>pending-read-ops</td>
<td>--</td>
<td>Number of read requests received by the cluster and waiting to be handled.</td>
</tr>
<tr>
<td>pending-read-repair-tasks</td>
<td>--</td>
<td>Number of read repair operations in the queue waiting to run.</td>
</tr>
<tr>
<td>pending-repair-tasks</td>
<td>--</td>
<td>Manual repair tasks pending, operations to be completed during anti-entropy repair of a node.</td>
</tr>
<tr>
<td>pending-repl-on-write-tasks</td>
<td>--</td>
<td>Pending tasks related replication of data after an insert or update to a row.</td>
</tr>
<tr>
<td>pending-request-responses</td>
<td>--</td>
<td>Progress of streamed rows from the receiving node.</td>
</tr>
<tr>
<td>pending-streams</td>
<td>--</td>
<td>Progress of streamed rows from the sending node.</td>
</tr>
<tr>
<td>pending-write-ops</td>
<td>--</td>
<td>Number of write requests received by the cluster and waiting to be handled.</td>
</tr>
<tr>
<td>read-latency-op</td>
<td>microseconds</td>
<td>Average response time to a client read request.</td>
</tr>
<tr>
<td>read-ops</td>
<td>--</td>
<td>The number of read requests per second.</td>
</tr>
<tr>
<td>write-latency-op</td>
<td>microseconds</td>
<td>The average response time to a client write request.</td>
</tr>
<tr>
<td>write-ops</td>
<td>--</td>
<td>The write requests per second.</td>
</tr>
</tbody>
</table>

**Column Family Metrics Keys**

This list of keys corresponds to column family-specific metrics collected by OpsCenter:

<table>
<thead>
<tr>
<th>Key</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cf-keycache-hit-rate</td>
<td>%</td>
<td>Cache requests that resulted in a key cache hit.</td>
</tr>
<tr>
<td>cf-keycache-hits</td>
<td>--</td>
<td>Number of read requests that resulted in the requested row key being found in the key cache.</td>
</tr>
<tr>
<td>cf-keycache-requests</td>
<td>--</td>
<td>Total number of read requests on the key cache.</td>
</tr>
<tr>
<td>cf-live-disk-used</td>
<td>bytes</td>
<td>Disk space used by a column family for readable data.</td>
</tr>
<tr>
<td>cf-live-sstables</td>
<td>--</td>
<td>Current number of SSTables for a column family.</td>
</tr>
<tr>
<td>Metric</td>
<td>Unit</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cf-pending-tasks</td>
<td>--</td>
<td>Number of pending reads and writes on a column family.</td>
</tr>
<tr>
<td>cf-read-latency-op</td>
<td>microsecond</td>
<td>Internal response time to a successful request to read data from a column family.</td>
</tr>
<tr>
<td>cf-read-ops</td>
<td>--</td>
<td>Read requests per second on a column family.</td>
</tr>
<tr>
<td>cf-rowcache-hit-rate</td>
<td>--</td>
<td>Percentage of cache requests that resulted in a row cache hit.</td>
</tr>
<tr>
<td>cf-rowcache-hits</td>
<td>--</td>
<td>Number of read requests on the row cache.</td>
</tr>
<tr>
<td>cf-rowcache-requests</td>
<td>--</td>
<td>Total number of read requests on the row cache.</td>
</tr>
<tr>
<td>cf-total-disk-used</td>
<td>--</td>
<td>Disk space used by a column family for live or old data (not live).</td>
</tr>
<tr>
<td>cf-write-latency-op</td>
<td>microsecond</td>
<td>Internal response time to a successful request to write data to a column family.</td>
</tr>
<tr>
<td>cf-write-ops</td>
<td>--</td>
<td>Write requests per second on a column family.</td>
</tr>
</tbody>
</table>

**Operating System Metrics Keys**

This list of keys corresponds to operating system (OS) metrics collected by OpsCenter:

<table>
<thead>
<tr>
<th>Key</th>
<th>OS</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap-committed</td>
<td>all*</td>
<td>bytes</td>
<td>Allocated memory guaranteed for the Java heap.</td>
</tr>
<tr>
<td>heap-max</td>
<td>all*</td>
<td>bytes</td>
<td>Maximum amount that the Java heap can grow.</td>
</tr>
<tr>
<td>heap-used</td>
<td>all*</td>
<td>bytes</td>
<td>Average amount of Java heap memory used by Cassandra processes.</td>
</tr>
<tr>
<td>nonheap-committed</td>
<td>all*</td>
<td>bytes</td>
<td>Allocated memory, guaranteed for Java nonheap.</td>
</tr>
<tr>
<td>nonheap-max</td>
<td>all*</td>
<td>bytes</td>
<td>Maximum amount that the Java nonheap can grow.</td>
</tr>
<tr>
<td>nonheap-used</td>
<td>all*</td>
<td>bytes</td>
<td>Average amount of Java nonheap memory used by Cassandra processes.</td>
</tr>
<tr>
<td>os-cpu-idle</td>
<td>all*</td>
<td>%</td>
<td>Time the CPU is idle.</td>
</tr>
<tr>
<td>os-cpu-iowait</td>
<td>Linux</td>
<td>%</td>
<td>Time the CPU devotes to waiting for I/O to complete.</td>
</tr>
<tr>
<td>os-cpu-nice</td>
<td>Linux</td>
<td>%</td>
<td>Time the CPU devotes to processing nice tasks.</td>
</tr>
<tr>
<td>os-cpu-privileged</td>
<td>Windows</td>
<td>%</td>
<td>Time the CPU devotes to processing privileged instructions.</td>
</tr>
<tr>
<td>os-cpu-steal</td>
<td>Linux</td>
<td>%</td>
<td>Time the CPU devotes to tasks stolen by virtual operating systems.</td>
</tr>
<tr>
<td>os-cpu-system</td>
<td>Linux, OSX</td>
<td>%</td>
<td>Time the CPU devotes to system processes.</td>
</tr>
<tr>
<td>os-cpu-user</td>
<td>all*</td>
<td>%</td>
<td>Time the CPU devotes to user processes.</td>
</tr>
<tr>
<td>os-disk-await</td>
<td>Linux, Windows</td>
<td>MS</td>
<td>Average completion time of each request to the disk.</td>
</tr>
<tr>
<td>os-disk-free</td>
<td>all*</td>
<td>GB</td>
<td>Free space on a specific disk partition.</td>
</tr>
<tr>
<td>os-disk-queue-size</td>
<td>Linux, Windows</td>
<td>--</td>
<td>Average number of requests queued due to disk latency issues.</td>
</tr>
<tr>
<td>os-disk-read-rate</td>
<td>Linux, Windows</td>
<td>--</td>
<td>Rate of reads per second to the disk.</td>
</tr>
<tr>
<td>os-disk-read-throughput</td>
<td>Linux, Windows</td>
<td>mb/sec</td>
<td>Average disk throughput for read operations.</td>
</tr>
<tr>
<td>os-disk-request-size</td>
<td>Linux</td>
<td>sectors</td>
<td>Average size of read requests issued to the disk.</td>
</tr>
</tbody>
</table>
### Managing Events and Alerts

Using these methods, you can get information about log events, such as start-up and shutdown, and configure alert thresholds for a number of Cassandra metrics.

<table>
<thead>
<tr>
<th>Event and Alert Methods</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve Opscenter events.</td>
<td>GET /{cluster_id}/events</td>
</tr>
<tr>
<td>Retrieve configured alert rules.</td>
<td>GET /{cluster_id}/alert-rules</td>
</tr>
<tr>
<td>Retrieve a specific alert rule.</td>
<td>GET /{cluster_id}/alert-rules/{alert_id}</td>
</tr>
<tr>
<td>Create a new alert rule.</td>
<td>POST /{cluster_id}/alert-rules/</td>
</tr>
<tr>
<td>Update an alert rule.</td>
<td>PUT /{cluster_id}/alert-rules/{alert_id}</td>
</tr>
</tbody>
</table>

- all means Linux, OSX, and Windows operating systems.
Delete an alert rule.

DELETE /{cluster_id}/alert-rules/{alert_id}

Retrieve active alerts.

GET /{cluster_id}/alerts/fired

**HTTP Methods**

**GET/cluster_id/events**

Retrieve events created by OpsCenter.

**Path**  
cluster_id -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.

**Query params:**

- **count** -- The number of events to return. Defaults to 10.
- **timestamp** -- A timestamp specifying the point in time to start retrieving events. Specified as a unix timestamp in microseconds. Defaults to the current time.
- **reverse** -- A boolean (0 or 1) indicating whether to retrieve events in reverse order. Defaults to 1 (true). Events are retrieved starting from the time specified by the timestamp and going backward in time until 'count' events are found or there are no more events to retrieve.

Returns a list of dictionaries where each dictionary represents an event. An event dictionary contains properties describing that event.

**Example**

curl
http://127.0.0.1:8888/Test_Cluster/events?count=1
-X GET

**Output:**

```
{
   "action": 18,
   "api_source_ip": null,
   "event_source": "OpsCenter",
   "level": 1,
   "level_str": "INFO",
   "message": "OpsCenter starting up.",
   "source_node": null,
   "success": null,
   "target_node": null,
   "time": "1334768517145625",
   "user": null
}
```

**GET/cluster_id/alert-rules**

Retrieve a list of configured alert rules in OpsCenter.

**Path**  
cluster_id -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.

Returns a list of **AlertRule** objects.

**AlertRule**

```
{
   "id": <value>,
   "type": <value>,
   "threshold": <value>,
```

OpsCenter API

80
This table describes the property values of an AlertRule object:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>A unique ID that references an alert rule. Use only for retrieving alert rules.</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>The event or metric aggregation that triggers an alert. Values are currently limited to node-down or rolling-avg and are not editable.</td>
</tr>
<tr>
<td>threshold</td>
<td>Float</td>
<td>The metric boundary that triggers an alert when the threshold is crossed. Applicable only when type = rolling-avg.</td>
</tr>
<tr>
<td>comparator</td>
<td>String</td>
<td>Optional. Values are &lt; or &gt;.</td>
</tr>
<tr>
<td>duration</td>
<td>Int</td>
<td>How long (in minutes) the problem continues before firing the alert.</td>
</tr>
<tr>
<td>notify_interval</td>
<td>Int</td>
<td>How often (in minutes) to repeat the alert. Use 0 for a single notification.</td>
</tr>
<tr>
<td>enabled</td>
<td>Int</td>
<td>The state of the alert. Values are 0 (disabled) or 1 (activated).</td>
</tr>
<tr>
<td>metric</td>
<td>String</td>
<td>A key from list of metrics. Triggers an alert only if type = rolling-avg.</td>
</tr>
<tr>
<td>cf</td>
<td>String</td>
<td>Optional. The column family to monitor if the metric property is one of the Column Family Metrics Keys.</td>
</tr>
<tr>
<td>item</td>
<td>String</td>
<td>Optional. The device to monitor if the metric is one of the Operating System Metrics Keys.</td>
</tr>
<tr>
<td>dc</td>
<td>String</td>
<td>Optional. The name of the data center that contains nodes to be monitored.</td>
</tr>
</tbody>
</table>

Example

curl
  http://127.0.0.1:8888/Test_Cluster/alert-rules
  -X GET

Output:

```json
{
  "comparator": ">",
  "dc": ",
  "duration": 1.0,
  "enabled": 1,
  "id": "e0c356c7-62ff-4aa8-9b17-e305f101b69a",
  "metric": "write-ops",
  "notify_interval": 1.0,
  "threshold": 0.0,
  "type": "rolling-avg"
}
```

GET/cluster_id/alert-rules/alert_id
Retrieve a specific alert rule.

**Path arguments:**
- `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- `alert_id` -- A UUID that identifies a specific alert rule and has the value of an id property returned by GET /{cluster_id}/alert-rules.

Returns an AlertRule.

**Example**

```bash
curl http://127.0.0.1:8888/Test_Cluster/alert-rules/e0c356c7-62ff-4aa8-9b17-e305f101b69a
-X GET
```

**Output:**

```
{
    "comparator": ">",
    "dc": "",
    "duration": 1.0,
    "enabled": 1,
    "id": "e0c356c7-62ff-4aa8-9b17-e305f101b69a",
    "metric": "write-ops",
    "notify_interval": 1.0,
    "threshold": 0.0,
    "type": "rolling-avg"
}
```

**POST/cluster_id/alert-rules**

Create a new alert rule.

**Path arguments:**
- `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.

**Body:** A dictionary in the format of AlertRule describing the alert to create.

**Responses:**
- `201` -- Alert rule was created successfully

Returns the ID of the newly created alert.

**Example:**

```bash
curl http://127.0.0.1:8888/Test_Cluster/alert-rules
-X POST
-d '{
    "comparator": ">",
    "dc": "",
    "duration": 60.0,
    "enabled": 1,
    "metric": "heap-used",
    "notify_interval": 5.0,
    "threshold": 6291456000.0,
    "type": "rolling-avg"
}'
```

**Output:**

```
b375fd3e-3908-4be5-ae37-d8f3b8699a9f
```
PUT/cluster_id/alert-rules/alert_id
Update an existing alert rule.

Path arguments:
- cluster_id -- A key, which identifies the node’s cluster, in the dictionary returned by GET /cluster-configs.
- alert_id -- A UUID that identifies a specific alert rule and has the value of an id property returned by GET /{cluster_id}/alert-rules.

Body: A dictionary in the format of AlertRule describing the alert to create.

Responses:
- 200 -- Alert rule updated successfully

Example:
```
curl http://127.0.0.1:8888/Test_Cluster/alert-rules/b375fd3e-3908-4be5-ae37-d8f3b8699a9f -X PUT -d '{
  "duration": 120.0
}'
```

DELETE/cluster_id/alert-rules/alert_id
Delete an existing alert rule.

Path arguments:
- cluster_id -- A key, which identifies the node’s cluster, in the dictionary returned by GET /cluster-configs.
- alert_id -- A UUID that identifies a specific alert rule and has the value of an id property returned by GET /{cluster_id}/alert-rules.

Responses:
- 200 -- Alert rule removed successfully

Example:
```
curl http://127.0.0.1:8888/Test_Cluster/alert-rules/b375fd3e-3908-4be5-ae37-d8f3b8699a9f -X DELETE
```

GET/cluster_id/alerts/fired
Get all active alerts.

Path arguments:
- cluster_id -- A key, which identifies the node’s cluster, in the dictionary returned by GET /cluster-configs.

Returns a list of alerts that have been triggered. Each item in the list is a dictionary describing the triggered alert.

Example:
```
curl http://127.0.0.1:8888/Test_Cluster/alerts/fired -X GET
```

Output:
```
[
  {
    "alert_rule_id": "ca4cf071-03bd-486a-a8be-428e6cd7218a",
    "current_value": 31676303.333333332,
    "first_fired": 1336669233,
    "node": "10.11.12.150"
  }
]`
```
Managing Schedules and Backups

You can manage job schedules using these methods. Currently, the only type of job is a scheduled backup.

<table>
<thead>
<tr>
<th>Schedule Management Methods</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>List jobs scheduled to run in OpsCenter.</td>
<td>GET /{cluster_id}/job-schedules</td>
</tr>
<tr>
<td>Get the description of a scheduled job.</td>
<td>GET /{cluster_id}/job-schedules/{schedule_id}</td>
</tr>
<tr>
<td>Schedule a job.</td>
<td>POST /{cluster_id}/job-schedules</td>
</tr>
<tr>
<td>Modify a scheduled job.</td>
<td>PUT /{cluster_id}/job-schedules/{schedule_id}</td>
</tr>
<tr>
<td>Delete a scheduled job.</td>
<td>DELETE /{cluster_id}/job-schedules/{schedule_id}</td>
</tr>
</tbody>
</table>

Using these methods, get the status of and start backups across registered clusters.

<table>
<thead>
<tr>
<th>Backup Management Methods</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>List backups in the cluster.</td>
<td>GET /{cluster_id}/backups</td>
</tr>
<tr>
<td>List backups for a keyspace in the cluster.</td>
<td>GET /{cluster_id}/backups/{ks_name}</td>
</tr>
<tr>
<td>Run a backup.</td>
<td>POST /{cluster_id}/backups/run</td>
</tr>
</tbody>
</table>

HTTP Methods

GET/{cluster_id}/job-schedules
Retrieve a list of jobs scheduled to run in OpsCenter. Currently the only type of job is a scheduled backup.

Path  cluster_id  -- A key, which identifies the node's cluster, in the dictionary returned by arguments: GET /cluster-configs.
Responses: 201 -- Schedule created successfully

Returns a list of JobSchedule objects.

JobSchedule

```json
{
    "first_run_date": <value>,
    "first_run_time": <value>,
    "timezone": <value>,
    "interval": <value>,
    "interval_unit": <value>,
    "job_params": {<type>: <value>, <name>: <value>, ...},
    "id": <value>,
    "last_run": <value>
}
```

This table describes the property values of a JobSchedule object:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description of Values</th>
</tr>
</thead>
</table>

84
### OpsCenter API

**Path arguments:**

- **cluster_id** -- A key, which identifies the node's cluster, in the dictionary returned by `GET /cluster-configs`.

- **schedule_id** -- A unique ID of the scheduled job that matches the id attribute of a dictionary returned by `GET /{cluster_id}/job-schedules`.

**Returns a JobSchedule object**

**Example:**

```bash
curl 
http://127.0.0.1:8888/Test_Cluster/job-schedules/19119720-115a-4f2c-862f-e10e1fb90eed 
-X GET
```

**Example:**

```json
[
{
  "first_run_date": "2012-04-19",
  "first_run_time": "18:00:00",
  "id": "19119720-115a-4f2c-862f-e10e1fb90eed",
  "interval": 1,
  "interval_unit": "days",
  "job_params": {
    "cleanup_age": 30,
    "cleanup_age_unit": "days",
    "keyspaces": [],
    "type": "backup"
  },
  "last_run": "",
  "next_run": "2012-04-19 18:00:00 GMT",
  "timezone": "GMT"
},
...]
```

**GET/cluster_id/job-schedules/schedule_id**

Get the description of a specific scheduled job.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_run_date</td>
<td>String</td>
<td>A date in YYYY-MM-DD format the date for running the job.</td>
</tr>
<tr>
<td>first_run_time</td>
<td>String</td>
<td>A time in hh:mm:ss format specifying the time to begin running the job.</td>
</tr>
<tr>
<td>timezone</td>
<td>String</td>
<td>The time zone, listed in the OpsCenter /meta/timezones directory, for the job schedule. For example, GMT, US/Central, US/Pacific, and US/Eastern.</td>
</tr>
<tr>
<td>interval</td>
<td>Int</td>
<td>The number of times a job runs during an interval_unit. For example, an interval of 2 and an interval_unit of weeks runs a job every two weeks.</td>
</tr>
<tr>
<td>interval_unit</td>
<td>String</td>
<td>The unit of time of an interval. Values are minutes, hours, days, or weeks.</td>
</tr>
<tr>
<td>job_params</td>
<td>--</td>
<td>An object that describes the job. Only the type field is required and its value is currently limited to backup.</td>
</tr>
<tr>
<td>id</td>
<td>String</td>
<td>A unique ID that references a job schedule. Use only for retrieving a job schedule.</td>
</tr>
<tr>
<td>last_run</td>
<td>String</td>
<td>The date and time of the last successful run. Use only for retrieving a job schedule.</td>
</tr>
</tbody>
</table>
POST/cluster_id/job-schedules
Create a new scheduled job. You can create a scheduled job to run one time in the future by specifying an interval=-1 and interval_unit=null.

Path  cluster_id -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.

Body : A dictionary in the format of JobSchedule describing the scheduled job to create.

Responses:
201 -- Job schedule created successfully

Example

curl
http://127.0.0.1:8888/Test_Cluster/job-schedules/
-X POST
-d
'"first_run_date": "2012-04-19",
"first_run_time": "18:00:00",
"id": "19119720-115a-4f2c-862f-e10e1fb90eed",
"interval": 1,
"interval_unit": "days",
"job_params": {
  "cleanup_age": 30,
  "cleanup_age_unit": "days",
  "keyspaces": [],
  "type": "backup"
},
"last_run": "",
"next_run": "2012-04-19 18:00:00 GMT",
"timezone": "GMT"
}'

Output:
"905391b7-1920-486d-a633-282f22dce604"

PUT/cluster_id/job-schedules/schedule_id
Update an existing scheduled job.
Path arguments:

- `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- `schedule_id` -- A unique ID identifying the schedule job to update.

Body: A dictionary in the format of JobSchedule describing the scheduled job to create.

Responses:

- `200` -- Schedule updated successfully

Example

curl

```bash
http://127.0.0.1:8888/Test_Cluster/job-schedules/905391b7-1920-486d-a633-282f22dce604
-X PUT
-d
'"first_run_date": "2012-05-05",
 "first_run_time": "19:00:00"
'}

DELETE/cluster_id/job-schedules/schedule_id
Delete an existing scheduled job.

Path arguments:

- `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- `schedule_id` -- A unique ID identifying the schedule job to delete.

Responses:

- `200` -- Schedule deleted successfully

Example

curl

```bash
http://127.0.0.1:8888/Test_Cluster/job-schedules/905391b7-1920-486d-a633-282f22dce604
-X DELETE
```

GET/cluster_id/backups
Get a list of existing snapshots in the cluster.

Path arguments: `cluster_id` -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.

Returns a dictionary describing existing snapshots in the cluster. Each key in the dictionary is the ID of the snapshot and each value is a dictionary describing the snapshot.

Example

curl http://127.0.0.1:8888/Test_Cluster/backups
-X GET

Output:

```json
{
 "opscenter_adhoc_2012-04-19-17-41-12-UTC": {
 "id": "adhoc",
 "keyspaces": {
 "OpsCenter": {
 "nodes": [
 "127.0.0.1",
 "127.0.0.1"
 ],
 "size": 23180
 },
 "size": 23180
 }```
GET/cluster_id/backups/ks_name
Get a list of existing snapshots for a specific keyspace in the cluster.

Path arguments:
- cluster_id -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.
- ks_name -- The keyspace scheduled for snapshot.

Returns a dictionary describing existing snapshots in the cluster. Each key in the dictionary is the ID of the snapshot and each value is a dictionary describing the snapshot.

Example

curl http://127.0.0.1:8888/Test_Cluster/backups/movies_ks
-X GET

Output:

{
  "opscenter_adhoc_2012-04-15-06-47-UTC": {
    "id": "adhoc",
    "keyspaces": {
      "movies_ks": {
        "nodes": [
          "10.11.12.150",
          "10.11.12.152"
        ],
        "size": 14833
      }
    }
  },
  "time": 1334761607
}

POST/cluster_id/backups/run
Run a one time backup immediately.

Path arguments:
- cluster_id -- A key, which identifies the node's cluster, in the dictionary returned by GET /cluster-configs.

Body: A json dictionary containing the options to use for running this one-time backup.
Returns true to indicate a successful snapshot.

**Example**
Start a backup of the test keyspace on nodes 10.11.12.150 and 10.11.12.152 to run immediately.

```bash
curl http://127.0.0.1:8888/Test_Cluster/backups/run
-X POST
-d
'"keyspaces":
 {"test":
    {"nodes": ["10.11.12.150", "10.11.12.152"], "size": 29644}
  }
}'
```

Output:
The first true response indicates that the snapshot was successful for the specified node.

```
[ true,
  [ null,
    "10.11.12.150"
  ],
],
[ true,
  [ null,
    "10.11.12.152"
  ]
]
```

Note: You can schedule a one-time backup in the future using the `POST /{cluster_id}/job-schedules`. Specify an interval=-1 and interval_unit=null.

---

**Troubleshooting**

This section lists some common problems experienced with OpsCenter and solutions or workarounds.

**Cannot create a keyspace**

Due to a Python 2.6 or earlier bug, some users experience a problem adding a keyspace using Data Modeling OpsCenter features. OpsCenter cannot save a newly created keyspace.

Upgrading Python generally fixes this problem.

**Error exceptions. ImportError:libssl.so.0.9.8 displayed**

Your CentOS 5.x, Debian, OEL 5.5, RHEL 5.x, or Ubuntu operating system has OpenSSL 1.0.0 installed and you see an error containing this message.

exceptions.ImportError: libssl.so.0.9.8

To correct this situation, install OpenSSL 0.9.8.

- To install OpenSSL 0.9.8 on OEL and RHEL, run `sudo yum install openssl098`.
- To install OpenSSL 0.9.8 on CentOS, Debian, or Ubuntu, run `sudo apt-get install libssl0.9.8`.
OpsCenter agent port setting conflict

If you have a problem with OpsCenter, check for conflicts in port settings. The OpsCenter Agent uses port 7199 by default. If you have not changed the default port, check that Cassandra or another process on the node, is not set up to use this port.

If you set the OpsCenter Agent port to a host name instead of an IP address, the DNS provider must be online to resolve the host name. If the DNS provider is not online, intermittent problems should be expected.

Limiting the metrics collected by OpsCenter

If you have many column families, the number of metrics OpsCenter collects can become quite large. For information about how to reduce the number keyspaces or column families that are monitored, see Controlling data collection.

Java not installed or JAVA_HOME environment variable not set

If Java is not installed or if OpsCenter cannot find JAVA_HOME, you may see an error such as:

```
/usr/share/opscenter-agent/bin/opscenter-agent: line 98: exec: -X: invalid option
exec: usage: exec [-cl] [-a name] [command [arguments ...]] [redirection ...]
```

To correct this problem, install Java or set JAVA_HOME:

```
export JAVA_HOME=<path_to_java>
```

Insufficient user resource limits errors

Insufficient resource limits may result in an insufficient nofiles error:

```
2012-08-13 11:22:51-0400 [] INFO: Could not accept new connection (EMFILE)
```

See Recommended settings under Insufficient user resource limits errors in the Cassandra documentation.

Release notes

For information about new features and resolved issues in the following releases, see:

- OpsCenter 2.1.3
- OpsCenter 2.1.2
- OpsCenter 2.1
- OpsCenter 2.0
- OpsCenter 1.4.1
- OpsCenter 1.4
- OpsCenter 1.3.1
- OpsCenter 1.3

Before upgrading, check the compatibility of the OpsCenter version you want to run against your version of Cassandra or DataStax Enterprise.

OpsCenter 2.1.3

OpsCenter Enterprise Edition 2.1.3 supports Cassandra 1.2 when virtual nodes are disabled. This Edition also includes bug fixes.
What's new

OpsCenter Enterprise Edition 2.1.3 can monitor and administer Cassandra 1.2 as described in the documentation unless you enable virtual nodes. When you enable virtual nodes, OpsCenter chooses a single token for each node for operations such as collecting metrics. Attempting to move nodes, rebalance nodes, and perform other tasks involving token ranges is not supported.

Resolved issues

- Fixed issue processing keyspaces that only exist in 1 datacenter.
- Fixed data collection issues during compactions on secondary indexes.

OpsCenter 2.1.2

OpsCenter Enterprise Edition 2.1.2 changes include a new feature and bug fixes.

What's new

OpsCenter Enterprise Edition 2.1.2 can now work in multiple regions or IP forwarding deployments.

Resolved issues

- Better parsing of some device types when collecting metrics.
- Fixed ability to configure user access on tarball installations.
- Fixed Cluster Report feature on tarball installations.

OpsCenter 2.1

OpsCenter Enterprise Edition 2.1 key enhancements include data modeling and data browsing, report generation, and additional metrics for performance monitoring.

What's new

- Browsing data in the Cassandra database.
- Generation of a PDF report about the cluster.
- Capability to handle thousands of column families efficiently.
- Online feedback form.
- Optional HTTPS support.
- Installation of agents to RHEL and Debian clusters regardless of the location of the configuration file (opscenterd.conf).
- Truncation of column families. Truncation removes the data but not the column family itself.

OpsCenter 2.0

OpsCenter Enterprise Edition 2.0 is a major release that introduces backup and cluster enhancements.

What's new

- Multi-cluster management. You can manage multiple cluster from the same OpsCenter instance.
Release notes

- New Overview page for monitoring multiple clusters. It shows a condensed view of each cluster's Dashboard. It is visible only when multiple clusters exist.
- Perform and schedule operations from OpsCenter.
- Point OpsCenter at existing Cassandra clusters, where you can add, modify, or delete clusters from OpsCenter.
- Adds support and metrics for DataStax Enterprise 2.0 Search nodes.
- No longer compatible with Cassandra 0.7.

Resolved issues

- Fix opscenter-agent package not starting on machine restart.
- Allow node move operation when a keyspace is set to replicate to a non-existent data center.
- Fix agent listen address when SSL is disabled.
- Fix data explorer error when searching for row key containing "#".

OpsCenter 1.4.1

OpsCenter 1.4.1 is a patch release to OpsCenter 1.4. For upgrading instructions, follow the instructions for Upgrading OpsCenter.

Resolved issues

- Fix free-space disk check before moving node.
- Add additional support for Linux device types.
- Improvements to OpsCenter keyspace creation logic.
- Include heap options in tarball agent startup script.

OpsCenter 1.4

OpsCenter 1.4 adds support for two major operating systems and widens support for Python. For upgrading to 1.4, follow the instructions for Upgrading OpsCenter.

What's new

- Support for OpsCenter and Cassandra on Windows. Using a GUI installer, you can easily be up and running on a 32- or 64-bit Windows 7 or Windows Server 2008 in a few minutes. OpsCenter and OpsCenter agents run as Windows services.
- Support for OpsCenter on Mac OSX. You can install OpsCenter in a single directory using a tar package. Root permissions are not required.
- Extended support of Python. OpsCenter supports Python 2.5, 2.6, and 2.7 on Linux, Mac OSX, and Windows.

Resolved issues

- Detection of OpenSSL fails on RedHat Enterprise Linux (RHEL) 5. x.
- Configuring user access using the set_passwd.py utility fails.
- When you replace a node at position 0-1, OpsCenter obscures the new node in the ring diagram.
Release notes

- When viewing a table of a column family data in Data Explorer, the Row Key heading of the table is incorrectly labeled Super Column when you click the search button.
- When keys and super column names include spaces, the Data Explorer does not display data when you view all columns.
- The Performance graph of data over a period of time momentarily shows a false, steep drop in reads and writes.
- OpsCenter 1.2.3 on Red Hat Enterprise Linux behaves unpredictably when started with `/etc/init.d/opscenterd start`.

OpsCenter 1.3.1

OpsCenter 1.3.1 is a patch release to OpsCenter 1.3. For upgrades to 1.3.1, follow the instructions for Upgrading OpsCenter.

What's New

This release introduces support for OpsCenter on RedHat Enterprise Linux version 6.

Resolved Issues

- (DataStax Enterprise Only) Creating a new Keyspace while a rebalance is in progress causes the rebalance operation to error out.
- "Unable to import needed OpsCenter packages" error when running `set_passwd.py` to set up OpsCenter authentication.
- Data center and version information not returned from agents when `iostat` is missing on monitored nodes.

OpsCenter 1.3

OpsCenter 1.3 introduces the DataStax Enterprise (DSE) and DataStax Community (DSC) versions of OpsCenter. DSE contains enhanced features and functionality and is available to registered DataStax customers. DSE is a free version of OpsCenter and no longer requires registration with DataStax to download and use.

What's New

- (Enterprise-only) Auto-Rebalancing for Randomly Partitioned Clusters - OpsCenter Enterprise now has the ability to check if nodes are evenly distributed in each data center, and if not automatically moves tokens and data to balance the ring.
- (Enterprise-only) Proactive Alerts - OpsCenter Enterprise now has the ability to configure alert thresholds for commonly watched performance metrics and to send alerts when those thresholds are triggered.
- (Enterprise-only) The ability to track the progress of MapReduce jobs running on DSE Analytics nodes.
- More Reliable JMX Operations on Agent - Improved channels for pushing JMX operations to the agents from `opscenterd`, getting success or failure responses, and polling the status of an operation.
- Self-Contained Distribution - New tarball distribution of OpsCenter to allow for self-contained, non-root installations.

Resolved Issues

- Respect the user's JAVA_HOME and PATH environment when starting up `opscenter-agent`
- Sort nodes in correct token order in Cluster list view
- Cluster list view usability improvements
Limit how many files are rotated for agent.log
Use more precise timestamps for Events list
OS memory graph not graphing total correctly
Data Modeling - Fixed Create Keyspace using NetworkTopologyStrategy

Sending us feedback

Thanks for using OpsCenter Enterprise Edition. Please take a moment to let us know what you think. Start OpsCenter and click Feedback at the top of the console. The feedback form appears.

Enter your name, email address, and feedback. Click Send Feedback. We appreciate your comments.