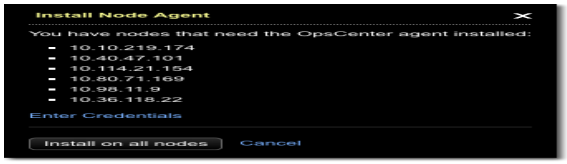
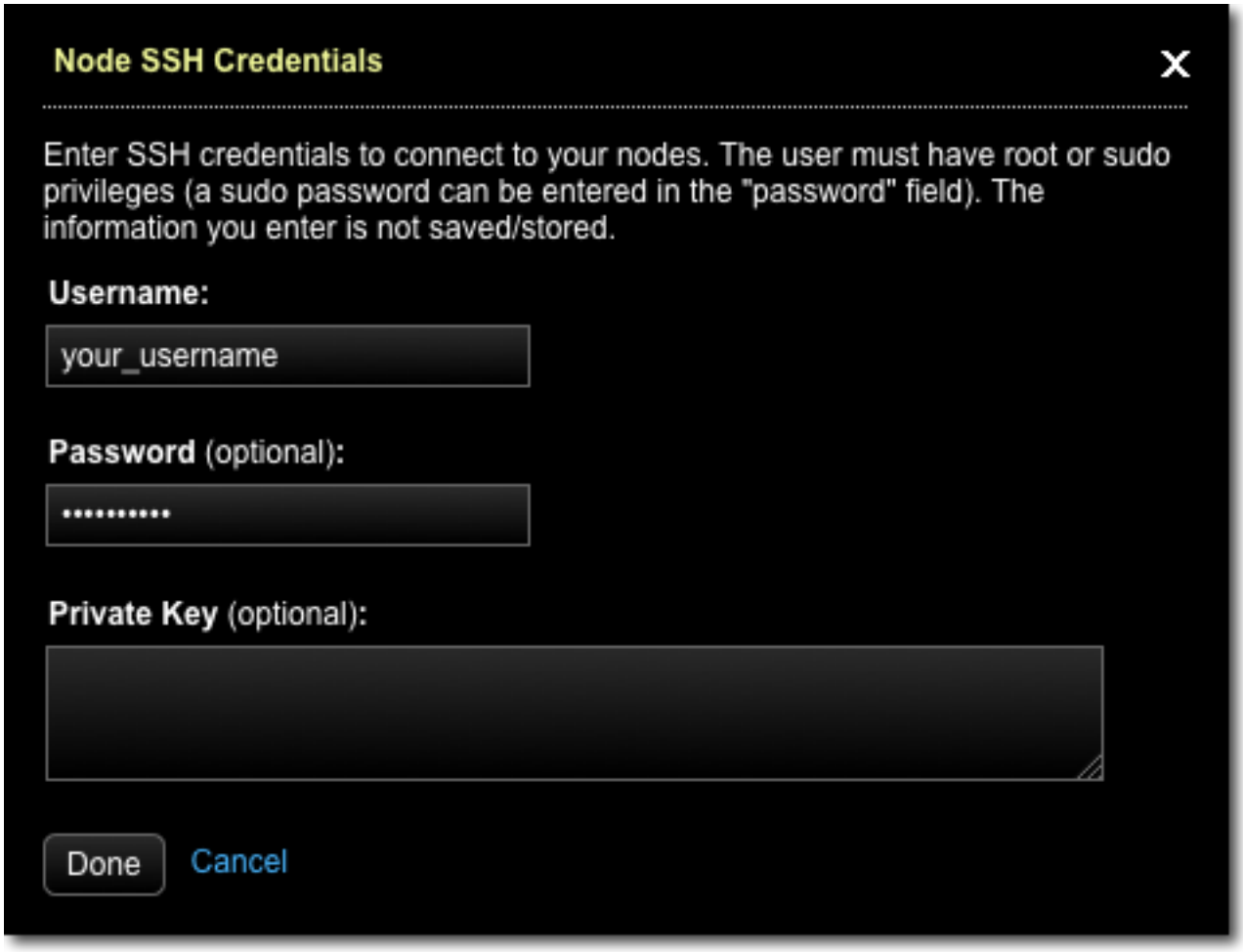
3. Enter the Hostnames or IP addresses of two or three nodes in the cluster and set the JMX set JMX and Thrift ports and credentials.
4. After connecting to the cluster, start installing the agents by clicking the **Fix** link located near the top of the Dashboard.



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

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>
   ```

   ### *Note*
   The agent.tar.gz file is not created until you run OpsCenter for the first time.

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/opscenter-agent` (in the background - default)

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

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 sudo 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:

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

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

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

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

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

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

4. If you **have** sudo 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:

      RHELCentOS: `sudo bin/install_agent.sh opscenter-agent.rpm <opscenter_host>`

      DebianUbuntu: `sudo 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** sudo 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`.

Admin role privileges

Users assigned the admin role can view all objects and perform all tasks. The following are tasks which may only be performed by an admin:

- Alerts
  - add
  - delete
  - modify
- Cluster operations
  - add nodes to a cluster
  - configure the cluster (all at once rather than a single node at a time)
  - rebalance
  - restart the cluster
- Column families
  - add column metadata
  - create
  - delete column metadata
  - delete index
  - drop
  - truncate
  - modify
- Keyspaces
  - create
  - drop
  - modify

- Node
    - cleanup
    - compact
    - configure
    - decommission
    - drain
    - flush
    - move
    - perform garbage collection
    - repair
    - restart
    - start
    - stop
- Install the OpsCenter agent on Cassandra nodes
- OpsCenter configuration
    - add an existing cluster to OpsCenter
    - delete a cluster from OpsCenter
    - edit the config for a cluster OpsCenter is monitoring
- Provisioning
    - add nodes to an existing cluster
    - provision a new cluster (local or EC2)
- Run a one-off backup
- Run a restore of a backup
- Scheduled backups
    - add
    - delete
    - modify

User role privileges

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:

/ersion	Operating System
0.9.8	CentOS 5.x, Debian, Mac OSX, Oracle Linux 5.5, RHEL 5.x, SuSe Enterprise 11.x, Ubuntu, and Windows

1.0.0	CentOS 6.x, Oracle Linux 6.1, and RHEL 6.x

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:

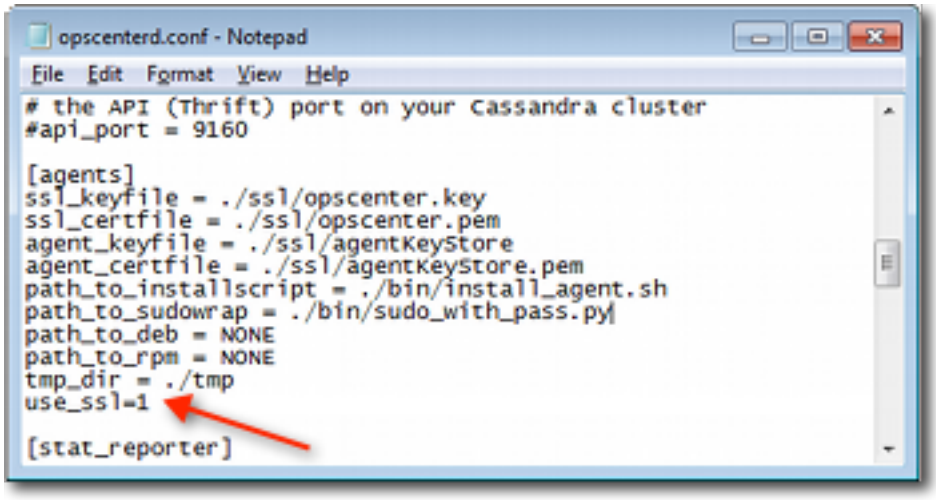
1. Go to the `opscenter\bin` directory:

   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.

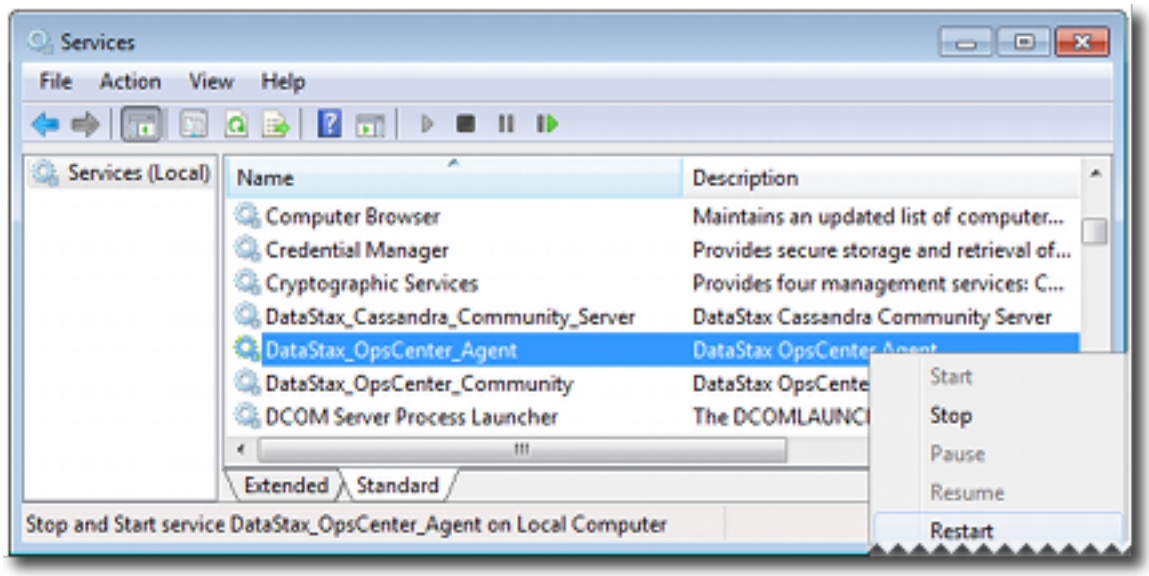
5. In the agents section, change `use_ssl` from `0` to `1` (or `true`), and then save the file.



6. Go to the opscenter\agent\conf directory:
   DataStax 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.

   ```
   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 and later can 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 used to identify the node. If broadcast_address is set in cassandra.yaml, this should be the same as that; otherwise, it is typically the same as listen_address in cassandra.yaml. A good check is to confirm that this address is the same as the address that nodetool ring outputs.

   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.

3. Repeat the above steps for each node.

   For example, here is the configuration for a three node cluster that spans two regions:

```
Region: us-west
Availability Zone: us-west-2

Node1
    public IP:  198.51.100.1
    private IP: 10.11.12.1
    Cassandra (cassandra.yaml)
        broadcast_address: 198.51.100.1
        listen_address:    10.11.12.1
    Agent (address.yaml)
        local_address:               198.51.100.1
        agent_rpc_interface:         10.11.12.1
        agent_rpc_broadcast_address: 198.51.100.1
    OpsCenter (opscenterd.conf)
        interface: 198.51.100.1

Node2
    public IP:  198.51.100.23
    private IP: 10.11.12.15
    Cassandra (cassandra.yaml)
        broadcast_address: 198.51.100.23
        listen_address:    10.11.12.15
    Agent (address.yaml)
        local_address:               198.51.100.23
        agent_rpc_interface:         10.11.12.15
        agent_rpc_broadcast_address: 198.51.100.23

Region: us-east
Availability Zone: us-east-1

Node1
    public IP:  203.0.113.20
    private IP: 10.11.13.28
    Cassandra (cassandra.yaml)
        broadcast_address: 203.0.113.20
        listen_address:    10.11.13.28
    Agent (address.yaml)
        local_address:               203.0.113.20
        agent_rpc_interface:         10.11.13.28
        agent_rpc_broadcast_address: 203.0.113.20
```

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:

Event	Code	Description
COMPACTION	0	Major compaction has occurred.
CLEANUP	1	Unused keys have been removed or cleaned up.

REPAIR	2	A repair operation has been initiated.
FLUSH	3	Memtables have been flushed to disk.
DRAIN	4	The commit log has been emptied, or drained.
DECOMMISSION	5	A leaving node has streamed its data to another node.
MOVE	6	Like NODE_MOVE; a new token range has been assigned.
NODE_DOWN	13	A node has stopped responding.
NODE_UP	14	An unresponsive node has recovered.
NODE_LEFT	15	A node has left, or been removed from, the ring.
NODE_JOIN	16	A node has joined the ring.
NODE_MOVE	17	A node has been assigned a new token range (the token has moved).
OPSC_UP	18	OpsCenter has been started and is operating.
OPSC_DOWN	19	OpsCenter was stopped or stopped running.
GC	20	Java garbage collection has been initiated.

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:

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

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,:file:*email.conf* looks like:

```
[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=CRITICAL,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
     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:

Number of days	Number of column families monitored	MB per node
31	5	200
31	10	300
31	20	500
365	5	250
365	10	380
365	20	630

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`):

```
[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:

In this 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).

1. Edit the `conf/clusters/<cluster>.conf` file.

2. Add the following time-to-live (`ttl`) values under a [cassandra_metrics] section:

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

3. Restart OpsCenter.

   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:

- `opscenterd.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 `opscenterd.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

By default, OpsCenter periodically sends usage metrics about the cluster to DataStax. This data is collected and reported when Opscenter starts up, and then every 24 hours afterwards. The goal is for this data to be completely anonymous. This includes not tracking or logging the ips that this data is sent from. You can turn this functionality completely off by adding the following lines to your opscenterd.conf file:

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

Here is a complete breakdown of the data OpsCenter will communicate back to DataStax. The data is sent in a key/value JSON format.

The following information is recorded about the OpsCenter install:

- 'install_id': This is a random uuid generated when OpsCenter starts for the first time. This is used for associating reports from the same install.
- 'is_paid': This is a flag indicating wether or not this is the free or enterprise version of OpsCenter.
- 'opscenter_version': The version of OpsCenter in use.
- 'opscenter_ram': The amount of RAM on the OpsCenter machine.
- 'opscenter_cores': The number of cores on the OpsCenter machine.
- 'opscenter_os': The generic name of the operating system of the OpsCenter machine. For example, linux/windows/mac.
- 'opscenter_os_sub': The specific name of the operating system of the OpsCenter machine. For example CentOS/Ubuntu/Debian.
- 'opscenter_os_version': The operating system version of the OpsCenter machine.
- 'opscenter_arch': The architecture of the OpsCenter machine.
- 'python_version': The version of python running on the OpsCenter machine.
- 'opscenter_instance_type': The instance type the OpsCenter machine, if OpsCenter is running in EC2.
- 'separate_storage': A flag indicating if OpsCenter is storing metrics in the cluster it is monitoring.
- 'config_diff': A list of the OpsCenter config options that were modified to be different than the defaults. This includes the names of the options that were changed but not the values of those options.

These stats are collected about each cluster OpsCenter is monitoring:

- 'cluster_id': An md5 hash of the cluster name. Used for identifying unique clusters while maintaining anonymity
- 'conf_id': An md5 hash of the file name the config for the cluster is stored in. Used for the same purposes as 'cluster_id'.
- 'partitioner': The partitioner the cluster is using.
- 'snitch': The snitch the cluster is using.
- 'keyspace_count': The number of keyspaces in the cluster.

- 'columnfamily_count': The number of column families in the cluster.
- 'strategy_options': A list of the replication options used for each keyspace in the cluster.
- 'cql3_cf_count': The number of column families created with CQL3 in the cluster.
- 'node_count': The number of nodes in the cluster.
- 'avg_token_count': The average number of tokens per node.
- 'cassandra_versions': A list of the different Cassandra versions in the cluster.
- 'bdp_version': A list of the different DataStax Enterprise versions in the cluster.
- 'rack_map': A map of each rack in the cluster and how many nodes are in that rack.
- 'dc_count': The number of datacenters in the cluster
- 'free_space': The amount of free disk space across the cluster.
- 'used_space': The amount of used disk space across the cluster.
- 'cluster_os': A list of the different operating systems used across the cluster.
- 'cluster_ram': The average amount of ram per node in the cluster.
- 'cluster_cores': The average number of cores per node in the cluster.
- 'cluster_instance_types': A list of the EC2 instance types in the cluster, if EC2 is being used.

[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.

[cassandra] install_location

The directory in which Cassandra is installed. If `install_location` is not specified, OpsCenter looks in the package-specific installation locations.

For a tarball installation of DataStax Enterprise, the `install_location` is `<dse_install_location>/resources/cassandra`.

[cassandra] conf_location

The location of the `cassandra.yaml` configuration file.

If `install_location` is specified, but `conf_location` is not, then `conf_location` is assumed to be `<install_location>/conf/cassandra.yaml`.

If `conf_location` is specified, it must be the absolute path to the Cassandra configuration file on all nodes. These settings are cluster-wide and require that the specified locations be correct for every node.

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.

2hr_ttl = 31536000

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. It runs on the client-side in a *web browser*.

Introduction to OpsCenter

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

- *Create Cluster*: create a cluster.
- *Add a cluster*: add 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.
- *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.
- Software update notification - be notified when an upgrade to OpsCenter or Cassandra or DSE on any of your clusters is available. The actual upgrade processes will be manual. When a new version is available a link at the top of the Dashboard appears. Clicking on it displays a dialog with a link to the new version of the software.
- *Cluster*: see your cluster from different perspectives and perform certain maintenance operations on cluster nodes.
- *Cluster administration*: add, modify, or remove a cluster from OpsCenter. Available in OpsCenter Enterprise Edition only.
- *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*: configure alert thresholds for a number of Cassandra cluster-wide, column family, and operating system metrics. Available in OpsCenter Enterprise Edition only.
- *Scehma*: create and manage keyspaces and the column families within them.
- *Data backups* - visually take, schedule, and manage backups across all registered clusters. Restore to clusters from backups. Available in OpsCenter Enterprise Edition only.
- *Data Explorer*: browse through column family data.
- Event Log: view the most recent OpsCenter log events, such as OpsCenter startup and shutdown.

Node administration

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

OpsCenter Enterprise Edition 3.0 can monitor and administer Cassandra 1.0 and later, but with Cassandra 1.2 it will not work as described in these procedures if 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:

Action	Description
View Metrics	Redirects you to the Performance area of OpsCenter where you can select metrics graphs and configure performance views for the selected node.
View Replication	Shows the replication relationships between the selected node and other nodes in the cluster. Visible in Ring View and Physical View.
Configure	Allows you to change the configuration of the node.
Start	Starts the DSE / Cassandra process on the node.
Stop	Stops the DSE / Cassandra process on the node.
Restart	Restarts the DSE / Cassandra process on the node.
Cleanup	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.
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.

Cluster administration

You can administer clusters from OpsCenter:

- *Creating a cluster*
- *Adding a cluster*
- *Generating a report*
- *Collecting diagnostic data*
- *Adding a node to a cluster*

- *Configuring a cluster*
- *Removing a cluster*
- *Rebalancing a cluster*
- *Restarting a cluster*
- *Modifying a cluster setting*

Note

OpsCenter Enterprise Edition 3.0 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 supports:

OpsCenter versions	Cassandra versions	DSE versions
1.4.x	0.7, 0.8, 1.0, 1.1	1.0, 2.0
2.0	0.8, 1.0, 1.1	1.0, 2.0
2.1	0.8, 1.0, 1.1	1.0, 2.0
3.0	1.0, 1.1, 1.2	1.0, 2.x, 3.0
3.1	1.0, 1.1, 1.2	1.0, 2.x, 3.0
3.2	1.1, 1.2	2.x, 3.0.x, 3.1.x

Creating a cluster

You can provision new clusters on colocated hosts or in the cloud.

To create a cluster:

1. Click **Create cluster**.
2. Fill out the form as appropriate.
3. Click **Save Cluster**.

Note

When deploying to EC2, the opscenterd to agent communication must be open to 0.0.0.0 in the EC2 security group (that is for ports 61620 and 61621).

Adding a cluster

To add a cluster to Opscenter:

1. Click **Add a cluster**.

2. Enter at least one hostname or IP address, newline-delimited, for the nodes comprising the cluster. OpsCenter discovers the other nodes for you.

   For example:

   ```
   ec2-123-45-6-789.us-west-1.compute.amazonaws.com
   ec2-234-56-7-890.us-west-1.compute.amazonaws.com
   ```

   Note
   It is recommended that you supply two or three hosts or IP addresses, in case a given node is down or decommissioned.

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**.

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
- Memory usage
- Operating system running on the node

You can save or print the PDF report.

Collecting diagnostic data

You can click **Diagnostics** to download a tarball that contains information about opscenterd and all nodes in a specified cluster. This can be used to attach to support tickets.

Adding a node to a cluster

To add a node to a cluster

1. Click **Add Node** in a cluster view.
2. Enter the following:

Option	Value
Package	The version of DSE to install on the node.
DataStax Credentials	The username and password you received when registering to Download DSE.
Nodes	The hostname or IP address, token, and software to install on the node (from Cassandra, Solr, and Hadoop). You can add more than one node by clicking **Add**.

Node Credentials (sudo)	The username and password to authenticate on the host. (Optional) the private SSH key to use to use for authentication.

3. Click **Add Nodes**.

Configuring a cluster

You can manage cassandra.yaml from OpsCenter. If the cluster exists in multiple datacenters, you can configure cassandra.yaml for a single datacenter, or for all nodes in a cluster. To manage cassandra.yaml for a single node by clicking on the **Actions** dropdown for a node.

To configure a cluster:

1. Click **Configure Cluster** in any of the Cluster views.
2. Edit the value for any of the options. For a description of the options in the file, see the documentation for the version of Cassandra or DSE which the cluster or node is running. For example, cassandra.yaml.
3. When finished editing, click **Save**.
4. You will be prompted to perform a rolling restart on the nodes in your cluster, or you may click **Cancel** if you do not wish to do a restart.

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**.

Note

When you delete a cluster, any EC2 nodes are not deleted.

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:
    - Calculates appropriate token ranges for each node and identify the nodes that need to move.
    - Makes sure that there is the appropriate free space to perform the rebalancing.
    - 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.
    - 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.

Restarting a node

You can start, stop, or restart the Cassandra or DSE service on any node. This can be done via the Actions dropdown on a node.

To restart a cluster:

1. In any cluster view, click on a node.
2. In the contextual menu select Restart from the **Actions** dropdown.

There is also rolling restart functionality for the entire cluster. This can be done via the "Restart Cluster" in any of the Cluster views.

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. (Optional) You can check thhe **DSE security (kerberos) is enabled on my cluster** and enter the service name.
5. (Optional) You can check the **Client node encryption is enabled on my cluster** and enter the **CA Certificate File Path**.
6. (Optional) You can check the **Validate SSL Certifcates** and enter the **Truststore File Path** and **Trustore Password**.

    For more information about enablinb Kerbeos see Security in the DSE Documentation.

7.Click **Save Cluster**.

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 3.0 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 and idea of system write workload and usage patterns.

Write Request Latency

The response time (in milliseconds) 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 and idea of system read workload and usage patterns.

Read Request Latency

The response time (in milliseconds) 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

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 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 3.0 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 milliseconds 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 milliseconds 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 actual 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 actual 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.

Disk Queue Size

Linux and Windows -- The average number of requests queued due to disk latency issues.

Mac OSX -- Not supported.

Disk Utilization

Linux and Windows -- The percentage of CPU time consumed by disk I/O.

Mac OSX -- 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.

Metric	Definition
Node Down	When a node is not responding to requests, it is marked as down.
Write Requests	The number of write requests per second. Monitoring the number of writes over a given time period can give you and idea of system write workload and usage patterns.
Write Request Latency	The response time (in milliseconds) 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.
Read Requests	The number of read requests per second. Monitoring the number of reads over a given time period can give you and idea of system read workload and usage patterns.
Read Request Latency	The response time (in milliseconds) 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.
CPU Usage	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.
Load	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.

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.

Metric	Definition

Heap Max	The maximum amount of shared memory allocated to the JVM heap for Cassandra processes.
Heap Used	The amount of shared memory in use by the JVM heap for Cassandra processes.
JVM CMS Collection Count	The number of concurrent mark-sweep (CMS) garbage collections performed by the JVM per second.
JVM ParNew Collection Count	The number of parallel new-generation garbage collections performed by the JVM per second.
JVM CMS Collection Time	The time spent collecting CMS garbage in milliseconds per second (ms/sec).
JVM ParNew Collection Time	The time spent performing ParNew garbage collections in ms/sec.
Data Size	The size of column family data (in gigabytes) that has been loaded/inserted into Cassandra, including any storage overhead and system metadata.
Compactions Pending	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.
Total Bytes Compacted	The number of sstable data compacted in bytes per second.
Total Compactions	The number of compactions (minor or major) performed per second.
Flush Sorter Tasks Pending	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).
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).
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).
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.
Internal Responses Pending	The number of pending tasks from various internal tasks such as nodes joining and leaving the 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. 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.
Memtable Post Flushers Pending	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).

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.
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).
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.
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).
Request Responses Pending	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.
Streams Pending	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.
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).

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.

Metric	Definition
Local Writes	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.
Local Write Latency	The response time in milliseconds for successful write operations on a column family. The time period starts when nodes receive a write request, and ends when nodes respond.
Local Reads	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.
Local Read Latency	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.
CF: KeyCache Hits	The number of read requests that resulted in the requested row key being found in the key cache.
CF: KeyCache Requests	The total number of read requests on the row key cache.

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

Metric	Definition
Memory Free	System memory that is not being used.
Memory Used	System memory used by application processes.

Memory Buffered	System memory used for caching file system metadata and tracking in-flight pages.
Memory Shared	System memory that is accessible to CPUs.
Memory Cached	System memory used by the OS disk cache.

CPU metrics on Linux

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

Disk metrics on Linux

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

Windows metrics

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

Memory metrics on Windows

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

CPU metrics on Windows

Metric	Definition
Idle	Percentage of time the CPU is idle.
Privileged	Percentage of time the CPU spends executing kernel commands.
User	Percentage of time allocated to user processes.

Disk metrics on Windows

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

Mac OSX metrics

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

Memory metrics on Mac OSX

Metric	Definition
Free Memory	System memory that is not being used.
Used Memory	System memory that is being used by application processes.

CPU metrics on Mac OSX

Metric	Definition
Idle	Percentage of time the CPU is idle.
System	Percentage of time allocated to system processes.
User	Percentage of time allocated to user processes

Disk metrics on Mac OSX

Metric	Definition
Disk Usage	Percentage of disk space Cassandra uses at a given time.
Free Space	Available disk space in GB.
Used Disk Space	Used disk space in GB.
Disk Throughput	Average disk throughput for read/write operations in megabytes per second. Exceptionally high disk throughput values may indicate I/O contention.

Managing backups and restoring from 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**.

Restoring from a backup

You can restore from any local backups that have been run by OpsCenter, but not from snapshots run from nodetool. You can pick any subset of column families that exist in the snapshot to restore.

To restore a backup:

1. Click **Data Backups**.
2. Find the backup you wish to restore in the list of backups.
3. Click **Restore**.
4. Choose which keyspaces to retore.
5. (Optional) Choose individual column families within a chosen keyspace to restore.
6. (Optional) Click the **Truncate/delete existing data before restore** checkbox.
7. (Optional) Click the **Throttle stream throughput at ____ MB** chekbox and enter the desired throttling value.
8. Click **Restore**.

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

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 **Schema** in the OpsCenter console.
2. Select **Add** in the Schema 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 **Schema** in the OpsCenter console.

2. From the list of keyspaces, select one of the keyspaces.

   In Keyspace 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** in the 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 Schema 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 **Schema** 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 **Schema** 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:

   - Add

     See *above*.

   - Delete

     Completely removes the column family from the keyspace. You may select more than one column family in a keyspace to delete.

   - 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

     Deletes all data from the column family but does not delete the column family itself. Removal of the data is irreversible.

4. When you select a column family, you see a list of manageable attributes: Properties (fully editable), Metadata (Add or Delete), and Secondary Indexes (Add or Delete).

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:

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 package installations*
- *Upgrading tarball installations*
- *Upgrading OpsCenter agents*

Before upgrading, check that the Cassandra or DataStax Enterprise version is compatible with the upgraded OpsCenter version:

OpsCenter Version	Cassandra Version	DataStax Enterprise Version
3.1	1.0, 1.1, 1.2	1.0, 2.0, 2.1, 2.2, 3.0.x
3.0	1.0, 1.1, 1.2	1.0, 2.0, 2.1, 2.2, 3.0
2.1.3	0.8, 1.0, 1.1, 1.2	1.0, 2.0, 2.1, 2.2
2.1	0.8, 1.0, 1.1	1.0, 2.0, 2.1, 2.2
2.0	0.8, 1.0, 1.1	1.0, 2.0
1.4.x	0.7, 0.8, 1.0, 1.1	1.0, 2.0

Upgrading package installations

Use the information in this section to upgrade to OpsCenter 3.0 from 2.1.x, 2.0, 1.4.x, or 1.3.x.

To upgrade to OpsCenter 3.0:

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 tarball installations

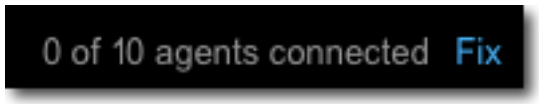
1. *Download* and extract the new tarball.
2. Copy the following files and directories from the old tarball installation directory to the new one:

   ```
   conf/clusters/*
   conf/event-plugins/*
   conf/install_id
   conf/log4j.properties
   conf/opscenterd.conf
   conf/ssl.conf
   ssl/*
   ```

3. *Stop* the opscenterd instance (if it is running) and start it from the new tarball installation directory.
4. Upgrade the agents either through the UI or by manually installing from the new tarballs.

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:



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

From tarballs

If the agents will be upgraded manually with tarballs, copy the new agent.tar.gz to all nodes, extract it, and copy the following files from the old agent tarball directories to the new ones:

```
conf/*
ssl/*
```

OpsCenter reference

This section contains miscellaneous information about OpsCenter.

OpsCenter and OpsCenter agent ports

The following table lists the default port numbers used by OpsCenter and the OpsCenter Agents:

Port	Description
OpsCenter ports	
8888	OpsCenter website. The opscenterd daemon listens on this port for HTTP requests coming directly from the browser. Configurable in `opscenterd.conf`.
50031	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)
61620	OpsCenter monitoring port. The opscenterd daemon listens on this port for TCP traffic coming from the agent.
OpsCenter agents ports (on the monitored nodes)	
7199	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.
8012	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.
9290	Hadoop Job Tracker Thrift port. The Job Tracker listens on this port for Thrift requests coming from the opscenterd daemon. (DataStax Enterprise only)
50030	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)
50060	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)
61621	OpsCenter agent port. The agents listen on this port for SSL traffic initiated by OpsCenter.
22	SSH port. Configurable in `opscenterd.conf`.
Solr Port and Demo applications port	
8983	Solr Port and Demo applications port.
Cassandra client port	
9160	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.

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:

- `/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)

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:

- *Authentication* describes how to authenticate when users and roles are configured.
- *OpsCenter Configuration* 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.
- *Managing Cluster Configurations* includes methods for reading and setting the cassandra.yaml for one or more nodes as well as setting the DSE node type.
- *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.
- *Snapshot Management and Restoring from Snapshots* describes methods for scheduling and running data snapshots and backups, getting information about existing snapshots, and restoring keyspaces and column families from snapshots.
- *Hadoop Status* shows how to get information about the current status of the Hadoop infrastructure in your DSE cluster.
- *Provisioning New Nodes and Clusters* details methods for creating new Cassandra and DSE nodes and clusters through OpsCenter, including launching Amazon EC2 instances to host the nodes.

This document contains the following sections:

Authentication

OpsCenter uses basic HTTP authentication when users and roles are configured. How your program authenticates depends on the library or tool you use. A example using cURL is provided.

Example:

```
curl http://127.0.0.1:8888/Test_Cluster/cluster
   -u <userid>:<password>
```

OpsCenter Configuration

You can manage the OpsCenter configuration for monitored clusters using these HTTP methods:

OpsCenter Configuration Management Methods	URL
List configuration information for all clusters.	`GET /cluster-configs`
List configuration information for a cluster.	`GET /cluster-configs/{cluster_id}`

Add a new cluster to OpsCenter.	`POST /cluster-configs`
Update a cluster configuration.	`PUT /cluster-configs/{cluster_id}`
Remove a cluster from OpsCenter management.	`DELETE /cluster-configs/{cluster_id}`

Getting Existing OpsCenter Cluster Configurations

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).

Cluster Config

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

Example:

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

Output:

```
{
  "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 arguments: **cluster_id** -- A **Cluster Config** ID.

Returns a **Cluster Config**.
Example:

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

Output:

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

Adding, Updating, and Removing Configured Clusters

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

Body : A configuration in the format of **Cluster Config**. A `seed_hosts` entry is required.

Responses: **201** -- Cluster was added successfully

Returns the ID of the new cluster.
Example:

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

Output:

```
"Test_Cluster"
```

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

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Body : A configuration in the same format as **Cluster Config**. A `seed_hosts` entry is required.

Responses: **204** -- Cluster updated successfully

Example:

```
curl -X PUT
  http://127.0.0.1:8888/cluster-configs/Test_Cluster
  -d '{
    "cassandra": {
      "seed_hosts": "192.168.1.1, 192.168.1.2"
    },
    "jmx": {
      "port": "7199"
    }
  }'
```

DELETE/cluster-configs/cluster_id

Remove a cluster from the list of those to be managed and monitored by OpsCenter.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Responses: **204** -- Cluster removed successfully

Example:

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

Managing Cluster Configurations

The following HTTP methods allow you to view and edit the cassandra.yaml configuration and set the DSE node type for some or all of the nodes in a cluster.

Cluster Configuration Management Methods	URL
Read the cassandra.yaml contents for a single node.	`GET /{cluster_id}/nodeconf/{node_ip}/`
Set the cassandra.yaml contents for a single node.	`POST /{cluster_id}/nodeconf/{node_ip}/`
Read the cassandra.yaml contents for one or all DCs.	`GET /{cluster_id}/clusterconf/{dc}/`
Set the cassandra.yaml contents for one or all DCs.	`POST /{cluster_id}/clusterconf/{dc}/`
Get the DSE role for a node.	`GET /{cluster_id}/dseconf/{node_ip}/nodety`
Set the DSE role for a node.	`POST /{cluster_id}/dseconf/{node_ip}/nodet`

cassandra.yaml Configuration

GET/cluster_id/nodeconf/node_ip/

Get the contents of the cassandra.yaml configuration file for a single node in the cluster.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- IP address of the target **Node**.

Format : If the *Accept* header is *text/yaml* or is not specified, the configuration will be returned as YAML. If the *Accept* type is *text/json*, the configuration will be returned as JSON.

Example:

```
curl http://127.0.0.1:8888/Test_Cluster/nodeconf/192.168.1.1
```

Output:

```
index_interval: 128
rpc_timeout_in_ms: 10000
compaction_throughput_mb_per_sec: 16
authority: org.apache.cassandra.auth.AllowAllAuthority
compaction_preheat_key_cache: true
row_cache_size_in_mb: 0
thrift_max_message_length_in_mb: 16
flush_largest_memtables_at: 0.75
in_memory_compaction_limit_in_mb: 64
rpc_server_type: sync
multithreaded_compaction: false
trickle_fsync: false
hinted_handoff_throttle_delay_in_ms: 1
row_cache_save_period: 0
hinted_handoff_enabled: true
cluster_name: Test Cluster
ssl_storage_port: 7001
populate_io_cache_on_flush: false
seed_provider:
- class_name: org.apache.cassandra.locator.SimpleSeedProvider
  parameters:
  - {seeds: 192.168.1.1, 192.168.1.2}
dynamic_snitch_reset_interval_in_ms: 600000
storage_port: 7000
request_scheduler: org.apache.cassandra.scheduler.NoScheduler
rpc_port: 9160
row_cache_provider: SerializingCacheProvider
reduce_cache_capacity_to: 0.6
saved_caches_directory: /var/lib/cassandra/saved_caches
max_hint_window_in_ms: 3600000
commitlog_sync: periodic
thrift_framed_transport_size_in_mb: 15
key_cache_save_period: 14400
authenticator: org.apache.cassandra.auth.AllowAllAuthenticator
dynamic_snitch_badness_threshold: 0.1
commitlog_directory: /var/lib/cassandra/commitlog
column_index_size_in_kb: 64
trickle_fsync_interval_in_kb: 10240
snapshot_before_compaction: false
concurrent_reads: 32
endpoint_snitch: org.apache.cassandra.locator.SimpleSnitch
encryption_options:
  keystore: conf/.keystore
  protocol: TLS
  algorithm: SunX509
  cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA]
  truststore_password: cassandra
  truststore: conf/.truststore
  internode_encryption: none
  keystore_password: cassandra
  store_type: JKS
concurrent_writes: 32
data_file_directories: [/var/lib/cassandra/data]
auto_snapshot: true
partitioner: org.apache.cassandra.dht.RandomPartitioner
dynamic_snitch_update_interval_in_ms: 100
```

```
commitlog_segment_size_in_mb: 32
rpc_keepalive: true
reduce_cache_sizes_at: 0.85
incremental_backups: false
commitlog_sync_period_in_ms: 10000
memtable_flush_queue_size: 4
rpc_address: 0.0.0.0
listen_address: 192.168.1.1
initial_token: null
```

JSON Format Example

```
curl
  -H 'Accept: text/json'
  http://127.0.0.1:8888/Test_Cluster/nodeconf/192.168.1.1
```

Output:

Cassandra Config

This Cassandra Config object is just a JSON representation of a cassandra.yaml file. The actual attributes and values will vary depending on what version of Cassandra or DSE is being used.


```
{
    "authenticator": "org.apache.cassandra.auth.AllowAllAuthenticator",
    "authority": "org.apache.cassandra.auth.AllowAllAuthority",
    "auto_snapshot": true,
    "cluster_name": "Test Cluster",
    "column_index_size_in_kb": 64,
    "commitlog_directory": "/var/lib/cassandra/commitlog",
    "commitlog_segment_size_in_mb": 32,
    "commitlog_sync": "periodic",
    "commitlog_sync_period_in_ms": 10000,
    "compaction_preheat_key_cache": true,
    "compaction_throughput_mb_per_sec": 16,
    "concurrent_reads": 32,
    "concurrent_writes": 32,
    "data_file_directories": [
        "/var/lib/cassandra/data"
    ],
    "dynamic_snitch_badness_threshold": 0.1,
    "dynamic_snitch_reset_interval_in_ms": 600000,
    "dynamic_snitch_update_interval_in_ms": 100,
    "encryption_options": {
        "algorithm": "SunX509",
        "cipher_suites": [
            "TLS_RSA_WITH_AES_128_CBC_SHA",
            "TLS_RSA_WITH_AES_256_CBC_SHA"
        ],
        "internode_encryption": "none",
        "keystore": "conf/.keystore",
        "keystore_password": "cassandra",
        "protocol": "TLS",
        "store_type": "JKS",
        "truststore": "conf/.truststore",
        "truststore_password": "cassandra"
    },
    "endpoint_snitch": "org.apache.cassandra.locator.SimpleSnitch",
```

```
        "flush_largest_memtables_at": 0.75,
        "hinted_handoff_enabled": true,
        "hinted_handoff_throttle_delay_in_ms": 1,
        "in_memory_compaction_limit_in_mb": 64,
        "incremental_backups": false,
        "index_interval": 128,
        "initial_token": null,
        "key_cache_save_period": 14400,
        "listen_address": "192.168.1.1",
        "max_hint_window_in_ms": 3600000,
        "memtable_flush_queue_size": 4,
        "multithreaded_compaction": false,
        "partitioner": "org.apache.cassandra.dht.RandomPartitioner",
        "populate_io_cache_on_flush": false,
        "reduce_cache_capacity_to": 0.6,
        "reduce_cache_sizes_at": 0.85,
        "request_scheduler": "org.apache.cassandra.scheduler.NoScheduler",
        "row_cache_provider": "SerializingCacheProvider",
        "row_cache_save_period": 0,
        "row_cache_size_in_mb": 0,
        "rpc_address": "0.0.0.0",
        "rpc_keepalive": true,
        "rpc_port": 9160,
        "rpc_server_type": "sync",
        "rpc_timeout_in_ms": 10000,
        "saved_caches_directory": "/var/lib/cassandra/saved_caches",
        "seed_provider": [
            {
                "class_name": "org.apache.cassandra.locator.SimpleSeedProvider",
                "parameters": [
                    {
                        "seeds": "192.168.1.1, 192.168.1.2"
                    }
                ]
            }
        ],
        "snapshot_before_compaction": false,
        "ssl_storage_port": 7001,
        "storage_port": 7000,
        "thrift_framed_transport_size_in_mb": 15,
        "thrift_max_message_length_in_mb": 16,
        "trickle_fsync": false,
        "trickle_fsync_interval_in_kb": 10240
    }
```


POST/cluster_id/nodeconf/node_ip/

Set the contents of the cassandra.yaml configuration file for a single node in the cluster.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- IP address of the target **Node**.

Body : A YAML or JSON representation of the cassandra.yaml configuration for the node. If a JSON format is used, the *Accept* header should be set to *text/json*.

Example:

```
curl
  -X POST
  http://127.0.0.1:8888/Test_Cluster/nodeconf/192.168.1.1
  -d
    'index_interval: 128
     rpc_timeout_in_ms: 10000
     compaction_throughput_mb_per_sec: 16
     authority: org.apache.cassandra.auth.AllowAllAuthority
     compaction_preheat_key_cache: true
     row_cache_size_in_mb: 0
     thrift_max_message_length_in_mb: 16
     flush_largest_memtables_at: 0.75
     in_memory_compaction_limit_in_mb: 64
     rpc_server_type: sync
     multithreaded_compaction: false
     trickle_fsync: false
     hinted_handoff_throttle_delay_in_ms: 1
     row_cache_save_period: 0
     hinted_handoff_enabled: true
     cluster_name: Test Cluster
     ssl_storage_port: 7001
     populate_io_cache_on_flush: false
     seed_provider:
     - class_name: org.apache.cassandra.locator.SimpleSeedProvider
       parameters:
       - {seeds: 192.168.1.1, 192.168.1.2}
     dynamic_snitch_reset_interval_in_ms: 600000
     storage_port: 7000
     request_scheduler: org.apache.cassandra.scheduler.NoScheduler
     rpc_port: 9160
     row_cache_provider: SerializingCacheProvider
     reduce_cache_capacity_to: 0.6
     saved_caches_directory: /var/lib/cassandra/saved_caches
     max_hint_window_in_ms: 3600000
     commitlog_sync: periodic
     thrift_framed_transport_size_in_mb: 15
     key_cache_save_period: 14400
     authenticator: org.apache.cassandra.auth.AllowAllAuthenticator
     dynamic_snitch_badness_threshold: 0.1
     commitlog_directory: /var/lib/cassandra/commitlog
     column_index_size_in_kb: 64
     trickle_fsync_interval_in_kb: 10240
     snapshot_before_compaction: false
     concurrent_reads: 32
     endpoint_snitch: org.apache.cassandra.locator.SimpleSnitch
     encryption_options:
       keystore: conf/.keystore
       protocol: TLS
       algorithm: SunX509
       cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA]
       truststore_password: cassandra
       truststore: conf/.truststore
       internode_encryption: none
       keystore_password: cassandra
       store_type: JKS
     concurrent_writes: 32
```

```
    data_file_directories: [/var/lib/cassandra/data]
    auto_snapshot: true
    partitioner: org.apache.cassandra.dht.RandomPartitioner
    dynamic_snitch_update_interval_in_ms: 100
    commitlog_segment_size_in_mb: 32
    rpc_keepalive: true
    reduce_cache_sizes_at: 0.85
    incremental_backups: false
    commitlog_sync_period_in_ms: 10000
    memtable_flush_queue_size: 4
    rpc_address: 0.0.0.0
    listen_address: 192.168.1.1
    initial_token: null'
```

GET/cluster_id/nodeconf/dc/

Similar to `GET /{cluster_id}/nodeconf/{node_ip}/`, but collects the configuration for all nodes in one or all datacenters. Properties that are specific to a single node, such as listen_address, rpc_address, and broadcast_address will *not* be included in the returned config.

An error message is returned if properties that are not specific to a single node differ from one node to another in a cluster.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **dc** -- The name of a datacenter to collect configs for. Use the special value `all` to collect configs from all datacenters.

Query params: **allow_down_nodes** -- "0" to require all specified nodes to be up. "1" to return the conf from only the nodes that are available. Defaults to "1"

Format : If the *Accept* header is *text/yaml* or is not specified, the configuration will be returned as YAML. If the *Accept* type is *text/json*, the configuration will be returned as JSON.

POST/cluster_id/clusterconf/dc/

Similar to `POST /{cluster_id}/nodeconf/{node_ip}/`, but sets the configuration for all nodes in one or all datacenters. Properties that are specific to a single node, such as listen_address, rpc_address, and broadcast_address, will be ignored.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **dc** -- The name of a datacenter to configure. Use the special value `all` to set configs in all datacenters.

Body : A YAML or JSON representation of the cassandra.yaml configuration for the nodes. If a JSON format is used, the *Accept* header should be set to *text/json*.

DSE Node Types

GET/cluster_id/dseconf/node_ip/nodetype

Gets the DSE role for a particular node, which indicates if the node is running Hadoop services, Solr services, or neither. Note that this method is currently only supported when DSE has been installed with debian or RPM packages.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- IP address of the target **Node**.

Returns a JSON string containing either "cassandra", "hadoop", or "solr".

Example

```
curl localhost:8888/Test_Cluster/dseconf/192.168.1.1/nodetype
```

Output:

```
"hadoop"
```

POST/cluster_id/dseconf/node_ip/nodetype

Sets the DSE role for a particular node, which controls whether the node runs Hadoop services, Solr services, or neither. Note that this method is currently only supported when DSE has been installed with debian or RPM packages.

Changes to the node's DSE role will only take effect after the node has been restarted, which this method does not do.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- IP address of the target **Node**.

Body : A JSON string containing either cassandra, hadoop, or solr.

Example

```
curl -X POST
  localhost:8888/Test_Cluster/dseconf/192.168.1.1/nodetype
  -d '"hadoop"'
```

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.

Cluster and Node Retrieval Methods	URL
Retrieve cluster configuration information.	`GET /{cluster_id}/cluster`
Retrieve information about a cluster property.	`GET /{cluster_id}/cluster/{property}`
Retrieve information about nodes in the cluster.	`GET /{cluster_id}/nodes`
Retrieve information about a specific node.	`GET /{cluster_id}/nodes/{node_ip}`
Retrieve information about the property of a node.	`GET /{cluster_id}/nodes/{node_ip}/{property`
Retrieve information about a property for all nodes.	`GET /{cluster_id}/nodes/all/{property}`
Retrieve cluster-wide storage capacity information.	`GET /{cluster_id}/storage-capacity`

Cluster Properties

GET/cluster_id/cluster

Retrieve information about the cluster.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Returns a Cluster object.

Cluster

```
{
   "endpoint_snitch": Full class name of snitch,
```

```
    "name": Name of the cluster,
    "partitioner": Full class name of partitioner,
    "status": Status if the cluster is being restarted
}
```

Example:

```
curl http://127.0.0.1:8888/Test_Cluster/cluster
```

Output:

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

GET/cluster_id/cluster/property

Retrieve a single cluster property.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **property** -- A property of **Cluster**.

Returns a string.

Example:

```
curl http://127.0.0.1:8888/Test_Cluster/cluster/partitioner
```

Output:

```
"org.apache.cassandra.dht.RandomPartitioner"
```

Node Properties

GET/cluster_id/nodes

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

Path arguments: **cluster_id** -- A **Cluster Config** ID.

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>,
   "node_version": <name:value>,
   "load": <value>,
   "data_held": <value>,
   "mode": <value>,
   "streaming": <name:value, name:value, . . .>,
   "task_progress": <name:value, name:value, . . .>,
   "last_seen": <value>,
   "num_procs": <value>,
```

```
    "rpc_ip": <value>,
    "dc": <value>,
    "rack": <value>,
    "network_interfaces": <value array>,
    "partitions": {
      "data": <value list>,
      "commitlog": <value>,
      "saved_caches": <value>,
      "other": <value list>
    },
    "devices": {
      "data": <value list>,
      "commitlog": <value>,
      "saved_caches": <value>,
      "other": <value list>
    },
    "os": <value>,
    "has_jna": <value>,
    "ec2": {
      "instance-id": <value>,
      "instance-type": <value>,
      "ami-id": <value>,
      "placement": <value>
    }
  }
```

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

Property	Description of Values
node_ip	IP address
node_name	Hostname
token	Token assignment (string). If the node has multiple tokens (part of vnode support in Cassandra 1.2 and later), this will be a single randomly selected token from the node's set of tokens.
node_version	**Dictionary of component versions:** • cassandra: Cassandra version • dse: DSE version • jobtracker: Job Tracker version • tasktracker: Task Tracker version • search: Solr version
load	OS load, which corresponds to the command, 1min avg from uptime
data_held	Amount of Cassandra data on the node (in bytes)
mode	Examples are "normal", "decommissioned", "leaving". Controlled by Cassandra.
streaming	Dictionary of active outbound streams in the form `{<destination_node_ip>: <progress>}`, where `<progress>` is a float between 0.0 and 1.0. For example, 0.8 means the stream is 80% complete.

task_progress	Dictionary of active tasks in the form `{<task-name-string>: <progress>}`. Example task names include "major-compaction" and "minor-compaction". `<progress>` is the same format as it is for streaming.
last_seen	0 if the node is up, a UNIX timestamp indicating when the node went down if it is currently down, or null if the node state is unknown
num_procs	Number of processor cores
rpc_ip	The rpc_address set in cassandra.yaml, which is usually the same as the node_ip property
dc	Name of node's data center
rack	Name of node's rack
network_interfaces	Array of network interface names
partitions	**Dictionary of partitions on the node grouped by category:** • data: List of partitions where Cassandra data is stored • commitlog: Partition where the commitlog resides • saved_caches: Partition where caches are saved • other: List of other partitions on the node that are not used by Cassandra
devices	**Dictionary of devices on the node grouped by category:** • data: List of devices where Cassandra data is stored • commitlog: Device where the commitlog resides • saved_caches: Device where caches are saved • other: List of other devices on the node that are not used by Cassandra
os	Name of the operating system that will contain "linux", "mac", or "windows"
has_jna	True if the node has JNA enabled, false otherwise
ec2	**Dictionary of ec2 information:** • instance_id: the EC2 instance ID • instance-type: the size and type of the node's EC2 instance • ami-id: what AMI was used for the instance • placement: what AWS region and availability zone the node is located in

Example:

```
curl http://127.0.0.1:8888/Test_Cluster/nodes
```

Output:

```
{
  "192.168.1.28": {
    "data_held": 53067368.0,
    "dc": "Cassandra",
    "devices": {
```

```
      "commitlog": "disk0",
      "data": [
        "disk0"
      ],
      "other": [],
      "saved_caches": "disk0"
    },
    "ec2": {
      "ami-id": "ami-82fa58eb",
      "instance-id": "i-1c264a66",
      "instance-type": "m1.large",
      "placement": "us-east-1d"
    },
    "has_jna": true,
    "last_seen": 0,
    "load": 0.20000000000000001,
    "mode": "normal",
    "network_interfaces": [
      "lo0",
      "en0",
      "en1"
    ],
    "node_ip": "192.168.1.28",
    "node_name": "cassandra08",
    "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",
      "data": [
        "/dev/disk0s2"
      ],
      "other": [],
      "saved_caches": "/dev/disk0s2"
    },
    "rack": "rack1",
    "rpc_ip": "192.168.1.28",
    "streaming": {
      "192.168.1.27": 0.23,
      "192.168.1.29": 0.97,
    },
    "task_progress": {
      "minor-compaction": 0.447
    },
    "token": "344787738101924880846628172923066645152"
  },
  "192.168.1.29": {
    ...
  }
```

```
  ...
}
```

GET/cluster_id/nodes/node_ip
Retrieve data for a specific node.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- IP address matching the node_ip for the **Node**.

Returns a **Node** object.

GET/cluster_id/nodes/node_ip/property
Retrieve a single property for a node or all nodes.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- IP address matching the node_ip for the **Node**.
- **property** -- A property name, such as token, for a **Node** object.

Returns a single property from a **Node** object.
Example:

```
curl http://127.0.0.1:8888/Test_Cluster/nodes/1.2.3.4/token
```

Output:

```
"344787738101924880846628172923066645152"
```

GET/cluster_id/nodes/all/property
Retrieve a single property for all nodes.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **property** -- A property name for a **Node** object, such as "token".

Returns a dictionary of `{<node_ip>: <property value>}` for all nodes in the cluster.
Example:

```
curl http://127.0.0.1:8888/Test_Cluster/nodes/all/token
```

Output:

```
{
  "1.2.3.3": "0",
  "1.2.3.4": "344787738101924880846628172923066645152",
  "1.2.3.5": "149812098999132048118520102380198341 27"
}
```

Storage Capacity

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

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Returns:

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

Example:

```
curl http://127.0.0.1:8888/Test_Cluster/storage-capacity
```

Output:

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

Performing Cluster Operations

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

Node Administration Methods	
Initiate JVM garbage collection on a node.	`POST /{cluster_id}/ops/gc/{node_ip}`
Assign a new token to the node.	`PUT /{cluster_id}/ops/move/{node_ip}`
Drain a node.	`PUT /{cluster_id}/ops/drain/{node_ip}`
Decommission a node.	`PUT /{cluster_id}/ops/decommission/{node_ip}`
Run a removetoken operation.	`PUT /{cluster_id}/ops/removetoken/{node_ip}`
Assassinate an endpoint.	`PUT /{cluster_id}/ops/assassinate/{node_ip}`
Clean up a keyspace.	`POST /{cluster_id}/ops/cleanup/{node_ip}/{ks`
Flush memtables from a keyspace.	`POST /{cluster_id}/ops/flush/{node_ip}/{ks_n`
Repair a keyspace.	`POST /{cluster_id}/ops/repair/{node_ip}/{ks_`
Compact a keyspace.	`POST /{cluster_id}/ops/compact/{node_ip}/{ks`
Process Management Methods	
Start Cassandra/DSE on a node	`POST /{cluster_id}/ops/start/{node_ip}`
Stop Cassandra/DSE on a node	`POST /{cluster_id}/ops/stop/{node_ip}`
Restart Cassandra/DSE on a node	`POST /{cluster_id}/ops/restart/{node_ip}`
Perform a rolling restart of the cluster	`POST /{cluster_id}/ops/restart`
Cluster Rebalancing Methods	
List moves to balance a cluster.	`GET /{cluster_id}/ops/rebalance`
Run a list of moves to balance a cluster.	`POST /{cluster_id}/ops/rebalance`
Request Management Methods	
Get the status of a long-running request.	`GET /{cluster_id}/request/{request_id}/statu`
Cancel a request.	`POST /{cluster_id}/request/{request_id}/canc`

Node Administration Methods

POST/cluster_id/ops/gc/node_ip
Initiate JVM garbage collection on a **Node**.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- IP address of the target **Node**.

Returns null.
Example:

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/ops/gc/1.2.3.4
```

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

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** to be assigned a new token.

Body : New token to assign to node.

Returns a **Request** ID.
Example:

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

Output:

```
"72ff69b2-9cf5-4777-a600-9173b3fe7e6a"
```

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

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** to be flushed of memtables.

Returns null.
Example:

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

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

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** to be decommissioned.

Returns null.
Example:

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

PUT/cluster_id/ops/removetoken/node_ip
Perform a removetoken operation on the given node.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** to run removetoken on.

Body : A JSON dictionary with one optional key, force, which accepts a boolean value. force may only be set to true when a removetoken operation is already in progress for the given node.

Returns a **Request** ID.
Example:

```
curl -X PUT
  http://127.0.0.1:8888/Test_Cluster/ops/removetoken/1.2.3.4
  -d '{"force": true}'
```

PUT/cluster_id/ops/assassinate/node_ip
Assassinates a node.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** to assassinate.

Returns a **Request** ID.
Example:

```
curl -X PUT
  http://127.0.0.1:8888/Test_Cluster/ops/assassinate/1.2.3.4
```

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

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **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. If empty, all column families will be cleaned up.

Returns null.
Example

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

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

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **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. If empty, all column families will be flushed.

Returns null.
Example

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

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

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** that initiates repair.
- **ks_name** -- Keyspace to be repaired.

Body : A JSON dictionary with three keys:

- **is_sequential: Required for cassandra 1.1 and up. Will throw an** error if used with earlier versions. Run the repair sequentially
- **is_local: Required for cassandra 1.2 and up. Will throw an error** if used with earlier versions. Use only nodes in the same data center during the repair
- **cfs: List of column families to repair. If this is empty, all** column families will be repaired.

Returns null.
Example

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

POST/cluster_id/ops/compact/node_ip/ks_name
Initiates a major compaction on a keyspace.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** that initiates the compaction.
- **ks_name** -- Keyspace to be compacted.

Body : List of column families to compact. If this is empty, all column families will be compacted.

Returns null.
Example

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

Process Management Methods

POST/cluster_id/ops/start/node_ip
Start the Cassandra/DSE process on a single node.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** to be started.

Returns a **Request** ID.
Example:

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/ops/start/10.11.12.72
```

Output:

```
"a34814a6-4896-11e2-a563-e0b9a54a6d93"
```

POST/cluster_id/ops/stop/node_ip

Stop the Cassandra/DSE process on a single node.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** to be stopped.

Returns a **Request** ID.
Example:

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/ops/stop/10.11.12.72
```

Output:

```
"c0d81d54-4896-11e2-a563-e0b9a54a6d93"
```

POST/cluster_id/ops/restart/node_ip

Restart the Cassandra/DSE process on a single node.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** to be restarted.

Returns a **Request** ID.
Example:

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/ops/restart/10.11.12.72
```

Output:

```
"e2212500-4896-11e2-a563-e0b9a54a6d93"
```

POST/cluster_id/ops/restart

Perform a rolling restart of the entire cluster or a select list of nodes.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **node_ip** -- **Node** to be restarted.

Body : A JSON dictionary with two optional keys:

- sleep: Amount of time in seconds to sleep between restarting each node. Defaults to 60.
- ips: A list of ips to restart. If left empty, all nodes will be restarted (this is the default behavior).

Returns a **Request** ID.
Example:

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/ops/restart/10.11.12.72
```

Output:

```
"e2212500-4896-11e2-a563-e0b9a54a6d93"
```

Cluster Rebalancing Methods

GET/cluster_id/ops/rebalance

Return a list of proposed moves to run to balance a cluster. Will throw an error if called on a cluster using vnodes

Path arguments: **cluster_id** -- A **Cluster Config** ID.

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
```

Output:

```
[
  [
    "850705917302346158658436518579420528 64",
    "10.11.12.152"
  ]
]
```

POST/cluster_id/ops/rebalance

Run the specified list of moves to balance a cluster. Will throw an error if called on a cluster using vnodes

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Opt. 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 -X POST
  http://127.0.0.1:8888/Test_Cluster/ops/rebalance
  -d
  '[
     [
       "850705917302346158658436518579420528 64",
       "10.11.12.152"
     ]
   ]'
```

Output:

```
"e330b179-1b9f-40c2-a2f5-d2f3d24aa85c"
```

Request Management Methods

Request

Requests are the method that OpsCenter uses to track potentially long-running requests that must be completed asynchronously. When these potentially long-running API calls are made, opscenterd will immediately return a Request ID that can be used to look up the status of the request.
Once a Request is started, you can fetch the status information for it until opscenterd is restarted or a large number of Requests have been started.
A Request status takes the following form:

```
{
   "id": ID,
   "state": STATE,
   "started": STARTED,
   "finished": FINISHED,
   "cluster_id": CLUSTER_ID,
   "details": DETAILS
}
```

Data:

- **ID** (*string*) -- The unique UUID for this Request. When an operation is potentially long-running, opscented will return this ID immediately.
- **STATE** (*string*) -- Either "running", "success", or "error"
- **STARTED** (*int*) -- A unix timestamp representing when the Request started
- **FINISHED** (*int*) -- A unix timestamp representing when the Request finished, or null if it has not finished yet
- **CLUSTER_ID** (*string*) -- The name of the cluster that the Request is operating on
- **DETAILS** -- Typically a string containing a status or error message, but may be a dictionary in the form `{<subrequest_id>: <Request>}` when the Request holds a collection of subrequests.

Content Types:

- JSON

GET/cluster_id/request/request_id/status

Check the status of an asynchronous request sent to OpsCenter.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **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
```

```
{
  "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. Not all requests can be cancelled.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **request_id** -- The ID returned by the API call that triggered the request.

Returns null.
Example

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

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:

	URL
Retrieve information about 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}/{attribu`
Add a keyspace to the cluster	`POST /{cluster_id}/keyspaces/{ks_name}`
Update an existing keyspace	`PUT /{cluster_id}/keyspaces/{ks_name}`
Drop a keyspace in a cluster	`DELETE /{cluster_id}/keyspaces/{ks_name}`
Column Family Management Methods	
Get the description of a column family	`GET /{cluster_id}/keyspaces/{ks_name}/cf/{cf_n`
Add a column family to a keyspace	`POST /{cluster_id}/keyspaces/{ks_name}/cf/{cf_`
Update attributes of a column family	`PUT /{cluster_id}/keyspaces/{ks_name}/cf/{cf_n`
Get the CQL3 CREATE query for a column family	`GET /{cluster_id}/keyspaces/{ks_name}/cf/{cf_n`
Set the data type for a column	`PUT /{cluster_id}/keyspaces/{ks_name}/cf/{cf_n`
Clear the data type for a column	`DELETE /{cluster_id}/keyspaces/{ks_name}/cf/{c`
Add an index to a column family	`POST /{cluster_id}/keyspaces/{ks_name}/cf/{cf_`
Drop an index from a column family	`DELETE /{cluster_id}/keyspaces/{ks_name}/cf/{c`
Truncate a column family	`DELETE /{cluster_id}/data/{ks_name}/{cf_name}`
Drop a column family itself	`DELETE /{cluster_id}/keyspaces/{ks_name}/cf/{c`

Keyspace Management Methods

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

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Opt. params:

- **ksfields** -- Comma delimited list of explicit keyspace properties to return.
- **cffields** -- 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
```

Output:

```
{
  "Keyspace1": {
    "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

Retrieve information about a specific keyspace in the cluster.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The value of a 'keyspace' property in the output of `GET /{cluster_id}/keyspaces`.

Opt. params:

- **ksfields** -- Comma delimited list of explicit keyspace properties to return.
- **cffields** -- 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
```

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 **Cluster Config** ID.
- **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/Keyspace1/replica_placement_strategy
```

Output:

```
"org.apache.cassandra.locator.SimpleStrategy"
```

POST/cluster_id/keyspaces/ks_name

Add a keyspace to the cluster.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **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: **201** -- 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:

Keyspace Storage Parameter	Keyspace Dictionary Key
strategy_class	replica_placement_strategy
strategy_options	strategy_options
durable_writes	durable_writes

Example

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

PUT/cluster_id/keyspaces/ks_name

Update a keyspace.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of the keyspace to update.

Body : A JSON dictionary of *all* keyspace attributes, not just those you wish to change.

The JSON body should be similar to the one used when creating a keyspace.

Example

```
curl -X PUT
  http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspace1
  -d '{
    "strategy_class": "org.apache.cassandra.locator.SimpleStrategy",
    "strategy_options": {"replication_factor": "2"},
    "durable_writes": true
  }'
```

DELETE/cluster_id/keyspaces/ks_name

Drop a keyspace from the cluster.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of the keyspace to delete.

Responses: **204** -- Keyspace deleted successfully

Example

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

Column Family Management Methods

GET/cluster_id/keyspaces/ks_name/cf/cf_name

Get the description of a column family.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **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
```

Output:

```
{
  "column_metadata": {
    "state": {
        "validation_class": "AsciiType",
        "index_type": null,
        "index_name": null,
        "index_options": null
    }
  },
  "column_type": "Standard",
```

```
  "min_compaction_threshold": 4,
  ...
}
```

POST/cluster_id/keyspaces/ks_name/cf/cf_name
Add a column family to a keyspace.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of an existing keyspace.
- **cf_name** -- The name of the column family to add.

Body : A JSON dictionary describing the attributes of the column family. Keys in the dictionary are valid column family attributes, such as column_type and comparator_type, in Cassandra.

Responses: **201** -- Column family created successfully

Example

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspace1/cf/MyColFam
  -d '{
    "column_type": "Standard",
    "comment": "",
    "comparator_type": "org.apache.cassandra.db.marshal.UTF8Type",
    "default_validation_class": "org.apache.cassandra.db.marshal.BytesType",
    "read_repair_chance": 1.0,
    "bloom_filter_fp_chance": 0.01,
    "subcomparator_type": null
  }'
```

PUT/cluster_id/keyspaces/ks_name/cf/cf_name
Update or modify select attributes of a column family.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of an existing keyspace.
- **cf_name** -- The name of an existing column family.

Body : A JSON dictionary describing the attributes of the column family to update. Keys in the dictionary are valid column family attributes, such as read_repair_chance and bloom_filter_fp_chance, in Cassandra. Note that Cassandra does not allow some attributes, such as the column type and the comparator type, to be altered once set.

Example

```
curl -X PUT
  http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspace1/cf/MyColFam
  -d '{
    "read_repair_chance": 0.1,
    "bloom_filter_fp_chance": 0.03,
  }'
```

GET/cluster_id/keyspaces/ks_name/cf/cf_name/create_query
Get the CQL3 CREATE query for a column family.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of an existing keyspace.
- **cf_name** -- The name of an existing column family.

Returns an array of CQL3 CREATE queries. Only column families that were created with CQL3 have CREATE queries. Any indexes that were created on the column family will also be returned.

Example

```
curl http://127.0.0.1:8888/Test_Cluster/keyspaces/test_ks/cf/users/create_query
```

Output:

```
[
    "CREATE TABLE test_ks.users (
      foo int PRIMARY KEY,
      bar int,
      baz int
    ) WITH
      bloom_filter_fp_chance=0.010000 AND
      caching='KEYS_ONLY' AND
      comment='' AND
      dclocal_read_repair_chance=0.000000 AND
      gc_grace_seconds=864000 AND
      read_repair_chance=0.100000 AND
      replicate_on_write='true' AND
      populate_io_cache_on_flush='false' AND
      compaction={'class': 'SizeTieredCompactionStrategy'} AND
      compression={'sstable_compression': 'SnappyCompressor'}",
    "CREATE INDEX users_bar_idx ON test_ks.users (bar)"
]
```

PUT/cluster_id/keyspaces/ks_name/cf/cf_name/column/colname

Set the validation class (data type) for a column.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of an existing keyspace.
- **cf_name** -- The name of an existing column family.
- **colname** -- The name of the column to set a validation class for.

Body : A JSON string of the new validation class, such as "UT8Type".

Responses: **200** -- Column validator was set successfully.

Returns null.

Example

```
curl -X PUT
  http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspace1/cf/Users/column/state
  -d '"AsciiType"'
```

DELETE/cluster_id/keyspaces/ks_name/cf/cf_name/column/colname

Clear the validation class (data type) for a column.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of an existing keyspace.
- **cf_name** -- The name of an existing column family.
- **colname** -- The name of the column that has the validation class to be cleared.

Responses: **200** -- Column validator was cleared successfully.

Returns null.

Example

```
curl -X DELETE
  http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspace1/cf/Users/column/state
```

POST/cluster_id/keyspaces/ks_name/cf/cf_name/index/colname

Add a secondary index to a column family.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of an existing keyspace.
- **cf_name** -- The name of an existing column family.
- **colname** -- The name of the column to be indexed.

Body : A JSON dictionary describing the attributes of the index. Valid keys include validation_class, index_type, index_name, and index_options.

Responses: **201** -- Index was added successfully

Returns null.

Example

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspace1/cf/Users/index/state
  -d '{
    "validation_class": "AsciiType",
    "index_type": "KEYS",
    "index_name": "state_index",
    "index_options": null
  }'
```

DELETE/cluster_id/keyspaces/ks_name/cf/cf_name/index/colname

Drop a secondary index from a column family.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of an existing keyspace.
- **cf_name** -- The name of an existing column family.
- **colname** -- The name of the indexed column.

Responses: **204** -- Index was dropped successfully

Returns null.

Example

```
curl -X DELETE
  http://127.0.0.1:8888/Test_Cluster/keyspaces/Keyspace1/cf/Users/index/state
```

DELETE/cluster_id/data/ks_name/cf_name

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

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The name of the keyspace containing the column family.
- **cf_name** -- The name of the column family to truncate.

Responses: **204** -- Column family truncated successfully

Returns null.

Example

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

DELETE/cluster_id/keyspaces/ks_name/cf/cf_name

Drop a column family.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **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 dropped successfully

Example

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

Retrieving Metric Data

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

Metric Retrieval Methods	URL
Retrieve cluster-wide metrics.	`GET /{cluster_id}/cluster-metrics/{dc}/{metric}`
Retrieve cluster-wide metrics about a device.	`GET /{cluster_id}/cluster-metrics/{dc}/{metric}/`
Retrieve cluster-wide metrics about a column family.	`GET /{cluster_id}/cluster-metrics/{dc}/{ks_name}`
Retrieve metrics about a node.	`GET /{cluster_id}/metrics/{node_ip}/{metric}`
Retrieve node-specific metrics about a device.	`GET /{cluster_id}/metrics/{node_ip}/{metric}/{de`
Retrieve node-specific metrics about a column family.	`GET /{cluster_id}/metrics/{node_ip}/{ks_name}/{c`

You can choose from a large number of *metric keys* 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:

Query Parameter	Description
start	(optional) A timestamp in seconds indicating the beginning of the time range to fetch. When omitted, this defaults to one day before the `end` parameter.
end	(optional) A timestamp in seconds indicating the end of the time range to fetch. When omitted, this defaults to the current time.

step	(optional) The resolution of the data points for the metric. Valid input options are: 1, 5, 120, or 1440 minutes; corresponding output intervals are 60, 200, 7200, or 86400 seconds. The default is a 1 minute step.
function	(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.

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. Data points are listed in chronological order, starting with the oldest data point first.

GET/cluster_id/cluster-metrics/dc/metric

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

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **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 -G
  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'
```

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
        ]
      ]
    }
}
```

GET/cluster_id/cluster-metrics/dc/metric/device

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 **Cluster Config** ID.
- **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, when requesting a disk metric, `all` will aggregate metrics from all disk devices.

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

Examples of Device Arguments

To determine the set of network interfaces that metrics are available for, you can run a query similar to the following:

```
curl http://localhost:8888/Test_Cluster/nodes/192.168.1.1/network_interfaces

["lo0", "eth0", "eth1"]
```

In this case, `lo0`, `eth0`, and `eth1` can all be used.
Disk devices can be discovered in a similar way.

```
curl http://localhost:8888/Test_Cluster/nodes/192.168.1.1/devices

{
  "commitlog": "sdb",
  "data": ["sda"],
  "saved_caches": "sda",
  "other": ["sdc"]
}
```

In this case, any of `sda`, `sdb`, or `sdc` may be used.
Finally, metrics are also captured for disk partitions and filesystems:

```
curl http://localhost:8888/Test_Cluster/nodes/192.168.1.1/partitions
```

```
{
  "commitlog": "/dev/sdb1",
  "data": ["/dev/sda1"],
  "saved_caches": "/dev/sda1",
  "other": ["/dev/sdc1"]
}
```

Here, the available partitions are `/dev/sda1`, `/dev/sdb1`, and `/dev/sdc1`. Keep in mind that you will need to URL-encode the items, so `/dev/sda1` will become `%2Fdev%2Fsda1`.
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.

```
curl -G
  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'
```

Output:

```
{
  "Total": {
    "AVERAGE": [
      [
        1334102400,
        null
      ],
      [
        1334188800,
        21.000694274902344
      ],
      [
        1334275200,
        8.736943244934082
      ],
      [
        1334361600,
        9.0
      ],
      [
        1334448000,
        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 **Cluster Config** ID.
- **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 Users column family in the Keyspace1 keyspace of the cluster over all data centers from May 1, 2012 00:00:00 to May 5, 2012 00:00:00 GMT:

```
curl -G
  http://127.0.0.1:8888/Test_Cluster/cluster-metrics/all/Keyspace1/Users/cf-live-disk-used
  -d 'function=max'
```

```
  -d 'start=1335830400'
  -d 'end=1336176000'
  -d 'step=1440'
```

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

```
{

  "Total": {
    "MAX": [
      [
        1335830400,
        9740462592.0
      ],
      [
        1335916800,
        9932527616.0
      ],
      [
        1336003200,
        null
      ],
      [
        1336089600,
        10644448512.0
      ]
    ]
  }
}
```

GET/cluster_id/metrics/node_ip/metric
Retrieve metric data for a single node.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **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 -G
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'
```

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 **Cluster Config** ID.
- **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:

```
curl -G
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'
```

Output:

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

```
{
  "/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 **Cluster Config** ID.
- **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 Users column family in the Keyspace1 keyspace by cluster node 10.11.12.150 from May 1, 2012 at 00:00:00 to May 5, 2012 00:00:00 GMT.

```
curl -G
http://127.0.0.1:8888/Test_Cluster/metrics/10.11.12.150/Keyspace1/Users/cf-write-latency-op
  -d 'function=max'
  -d 'start=1335830400'
  -d 'end=1336176000'
  -d 'step=1440'
```

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:

Key	Units	Description
data-load	bytes	Size of the data on the node.
pending-compaction-tasks	--	Number of compaction operations queued and waiting to run.
pending-flush-sorter-tasks	--	Number of pending tasks related to the first step in flushing memtables to disk as SSTables.
pending-flushes	--	Number of memtables queued for the flush process.
pending-gossip-tasks	--	Number of gossip messages and acknowledgments queued and waiting to be sent or received.
pending-hinted-handoff	--	Number of hints in the queue waiting to be delivered after a failed node comes up.
pending-internal-responses	--	Number of pending tasks from internal tasks, such as nodes joining and leaving the cluster.
pending-memtable-post-flushers	--	Number of pending tasks related to the last step in flushing memtables to disk as SSTables.
pending-migrations	--	Number of pending tasks from system methods that modified the schema.
pending-misc-tasks	--	Number of pending tasks from infrequently run operations, not measured by another metric.
pending-read-ops	--	Number of read requests received by the cluster and waiting to be handled.

pending-read-repair-tasks	--	Number of read repair operations in the queue waiting to run.
pending-repair-tasks	--	Manual repair tasks pending, operations to be completed during anti-entropy repair of a node.
pending-repl-on-write-tasks	--	Pending tasks related replication of data after an insert or update to a row.
pending-request-responses	--	Progress of streamed rows from the receiving node.
pending-streams	--	Progress of streamed rows from the sending node.
pending-write-ops	--	Number of write requests received by the cluster and waiting to be handled.
read-latency-op	microseconds	Average response time to a client read request.
read-ops	--	The number of read requests per second.
write-latency-op	microseconds	The average response time to a client write request.
write-ops	--	The write requests per second.
key-cache-hits	--	The number of key cache hits per second. (This metric is per-column family before Cassandra 1.1)
key-cache-requests	--	The number of key cache requests per second. (This metric is per-column family before Cassandra 1.1)
key-cache-hit-rate	%	The percentage of key cache lookups that resulted in a hit. (This metric is per-column family before Cassandra 1.1)
row-cache-hits	--	The number of row cache hits per second. (This metric is per-column family before Cassandra 1.1)
row-cache-requests	--	The number of row cache requests per second. (This metric is per-column family before Cassandra 1.1)
row-cache-hit-rate	%	The percentage of row cache lookups that resulted in a hit. (This metric is per-column family before Cassandra 1.1)
total-compactions-completed	--	Number of compaction tasks completed.
total-bytes-compacted	bytes	Number of bytes compacted per second.
cms-collection-count	--	Number of concurrent mark sweep garbage collections performed per second.
cms-collection-time	ms/sec	Average number of milliseconds spent performing CMS garbage collections per second.
par-new-collection-count	--	Number of ParNew garbage collections performed per second.
par-new-collection-time	ms/sec	Average number of milliseconds spent performing ParNew garbage collections per second.
heap-committed	bytes	Allocated memory guaranteed for the Java heap.
heap-max	bytes	Maximum amount that the Java heap can grow.
heap-used	bytes	Average amount of Java heap memory used.
nonheap-committed	bytes	Allocated memory, guaranteed for Java nonheap.
nonheap-max	bytes	Maximum amount that the Java nonheap can grow.
nonheap-used	bytes	Average amount of Java nonheap memory used.

Column Family Metrics Keys

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

Key	Units	Description
cf-keycache-hit-rate	%	Cache requests that resulted in a key cache hit. (This metric is global in Cassandra 1.1+.)
cf-keycache-hits	--	Number of read requests that resulted in the requested row key being found in the key cache. (This metric is global in Cassandra 1.1+.)
cf-keycache-requests	--	Total number of read requests on the key cache. (This metric is global in Cassandra 1.1+.)
cf-live-disk-used	bytes	Disk space used by a column family for readable data.
cf-live-sstables	--	Current number of SSTables for a column family.
cf-pending-tasks	--	Number of pending reads and writes on a column family.
cf-read-latency-op	microseconds	Internal response time to a successful request to read data from a column family.
cf-read-ops	--	Read requests per second on a column family.
cf-rowcache-hit-rate	--	Percentage of cache requests that resulted in a row cache hit. (This metric is global in Cassandra 1.1+.)
cf-rowcache-hits	--	Number of read requests on the row cache. (This metric is global in Cassandra 1.1+.)
cf-rowcache-requests	--	Total number of read requests on the row cache. (This metric is global in Cassandra 1.1+.)
cf-total-disk-used	--	Disk space used by a column family for live or old data (not live).
cf-write-latency-op	microseconds	Internal response time to a successful request to write data to a column family.
cf-write-ops	--	Write requests per second on a column family.
cf-bf-space-used	bytes	How large the bloom filter is.
cf-bf-false-positives	--	Number of bloom filter false positives per second.
cf-bf-false-ratio	%	Percentage of bloom filter lookups that resulted in a false positive.
solr-avg-time-per-req	milliseconds	Average time a search query takes in a DSE cluster using DSE search.
solr-errors	--	Errors per second that occur for a specific Solr core/index.
solr-requests	--	Requests per second made to a specific Solr core/index.
solr-timeouts	--	Timeouts per second on a specific Solr core/index.

Operating System Metrics Keys

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

Key	OS	Units	Description
os-cpu-idle	all*	%	Time the CPU is idle.
os-cpu-iowait	Linux	%	Time the CPU devotes to waiting for I/O to complete.
os-cpu-nice	Linux	%	Time the CPU devotes to processing nice tasks.
os-cpu-privileged	Windows	%	Time the CPU devotes to processing privileged instructions.

os-cpu-steal	Linux	%	Time the CPU devotes to tasks stolen by virtual operating systems.
os-cpu-system	Linux, OSX	%	Time the CPU devotes to system processes.
os-cpu-user	all*	%	Time the CPU devotes to user processes.
os-disk-await	Linux, Windows	MS	Average completion time of each request to the disk.
os-disk-free	all*	GB	Free space on a specific disk partition.
os-disk-queue-size	Linux, Windows	--	Average number of requests queued due to disk latency issues.
os-disk-read-rate	Linux, Windows	--	Rate of reads per second to the disk.
os-disk-read-throughput	Linux, Windows	mb/sec	Average disk throughput for read operations.
os-disk-request-size	Linux	sectors	Average size of read requests issued to the disk.
os-disk-request-size-kb	Windows	KB	Average size of read requests issued to the disk.
os-disk-throughput	OSX	mb/sec	Average disk throughput for read and write operations.
os-disk-usage	all*	%	Disk space used by Cassandra at a given time.
os-disk-used	all*	GB	Disk space used by Cassandra at a given time.
os-disk-utilization	Linux, Windows	%	CPU time consumed by disk I/O.
os-disk-write-rate	Linux, Windows	--	Rate of writes per second to the disk.
os-disk-write-throughput	Linux, Windows	mb/sec	Average disk throughput for write operations.
os-load	all*	--	Operating system load average
os-memory-avail	Windows	MB	Available physical memory.
os-memory-buffers	Linux	MB	Total system memory currently buffered.
os-memory-cached	Linux	MB	Total system memory currently cached.
os-memory-committed	Windows	MB	Memory in use by the operating system.
os-memory-free	Linux, OSX	MB	Total system memory currently free.
os-memory-pool-nonpaged	Windows	MB	Allocated pool-nonpaged memory.
os-memory-pool-paged	Windows	MB	Allocated pool-paged-resident memory.
os-memory-sys-cache-resident	Windows	MB	Memory used by the file cache.
os-memory-used	Linux, OSX	MB	Total system memory currently used.
os-net-received	all*	kb/sec	Speed of data received from the network.
os-net-sent	all*	kb/sec	Speed of data sent across the network.

- all means Linux, OSX, and Windows operating systems.

Managing Events and Alerts

Using these methods, you can get information about log events, such as node compactions and repairs triggered through OpsCenter, and configure alert thresholds for a number of Cassandra metrics.

Event and Alert Methods	URL
Retrieve OpsCenter events.	`GET /{cluster_id}/events`
Alert Methods	
Retrieve configured alert rules.	`GET /{cluster_id}/alert-rules`
Retrieve a specific alert rule.	`GET /{cluster_id}/alert-rules/{alert_id}`
Create a new alert rule.	`POST /{cluster_id}/alert-rules/`
Update an alert rule.	`PUT /{cluster_id}/alert-rules/{alert_id}`
Delete an alert rule.	`DELETE /{cluster_id}/alert-rules/{alert_id}`
Retrieve active alerts.	`GET /{cluster_id}/alerts/fired`

Event Methods

GET/cluster_id/events

Retrieve historical events logged by OpsCenter.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

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
```

Output:

```
{
  "action": 28,
  "api_source_ip": 192.168.1.12,
  "event_source": "OpsCenter",
  "level": 1,
  "level_str": "INFO",
  "message": "Restarting node 192.168.100.3",
  "source_node": 192.168.100.3,
  "success": null,
  "target_node": null,
  "time": "1334768517145625",
  "user": joe
}
```

Alert Methods

GET/cluster_id/alert-rules

Retrieve a list of configured alert rules in OpsCenter.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Returns a list of **AlertRule** objects.

AlertRule

```
{
  "id": <value>,
  "type": <value>,
  "threshold": <value>,
  "comparator": <value>,
  "duration": <value>,
  "notify_interval": <value>,
  "enabled": <value>,
  "metric": <value>,
  "cf": <value>,
  "item": <value>,
  "dc": <value>
}
```

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

Property	Type	Description of Values
id	String	A unique ID that references an alert rule. Use only for retrieving alert rules.
type	String	The event or metric aggregation that triggers an alert. Accepted values include rolling-avg, cluster-balance, and node-down. This field is not editable.
threshold	Float	The metric boundary that triggers an alert when the threshold is crossed. Applicable only when the type is rolling-avg.
comparator	String	Optional. Values are < or >.
duration	Int	How long (in minutes) the problem continues before firing the alert.
notify_interval	Int	How often (in minutes) to repeat the alert. Use *0* for a single notification.
enabled	Int	The state of the alert. Values are *0* (disabled) or *1* (enabled).
metric	String	A key from *list of metrics*. This field is only valid if the type is rolling-avg.
cf	String	Optional. The column family to monitor if the metric property is one of the *Column Family Metrics Keys*.
item	String	Optional. The device to monitor if the metric is one of the *Operating System Metrics Keys*.
dc	String	Optional. The name of the data center that contains nodes to be monitored. If omitted, all nodes will be monitored.

Example

```
curl http://127.0.0.1:8888/Test_Cluster/alert-rules
```

Output:

```
[
  {
```

```
    "comparator": ">",
    "dc": "us-east",
    "duration": 1.0,
    "enabled": 1,
    "id": "e0c356c7-62ff-4aa8-9b17-e305f101b69a",
    "metric": "write-latency",
    "notify_interval": 1.0,
    "threshold": 10000.0,
    "type": "rolling-avg"
  },
  ...
]
```

GET/cluster_id/alert-rules/alert_id

Retrieve a specific alert rule.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **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

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

Output:

```
{
  "comparator": ">",
  "dc": "us-east",
  "duration": 1.0,
  "enabled": 1,
  "id": "e0c356c7-62ff-4aa8-9b17-e305f101b69a",
  "metric": "write-latency",
  "notify_interval": 1.0,
  "threshold": 10000.0,
  "type": "rolling-avg"
}
```

POST/cluster_id/alert-rules

Create a new alert rule.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

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:

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/alert-rules
  -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 **Cluster Config** ID.
- **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 of fields from **AlertRule** to update.

Responses: **200** -- Alert rule updated successfully

Example:

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

DELETE/cluster_id/alert-rules/alert_id

Delete an existing alert rule.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **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 -X DELETE
  http://127.0.0.1:8888/Test_Cluster/alert-rules/b375fd3e-3908-4be5-ae37-d8f3b8699a9f
```

GET/cluster_id/alerts/fired

Get all alerts which are currently fired.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

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
```

Output:

```
[
  {
    "alert_rule_id": "ca4cf071-03bd-486a-a8be-428e6cd7218a",
    "current_value": 31676303.333333332,
```

```
      "first_fired": 1336669233,
      "node": "10.11.12.150"
    },
    {
      "alert_rule_id": "ca4cf071-03bd-486a-a8be-428e6cd7218a",
      "current_value": 28380117.5,
      "first_fired": 1336669233,
      "node": "10.11.12.152"
    }
  ]
```

Snapshot Management and Restoring from Snapshots

With these methods, you can schedule periodic snapshots, run snapshots immediately, view the snapshots that currently exist in the cluster, and restore from a snapshot.

Managing Snapshot Job Schedules	
List jobs scheduled to run in OpsCenter.	`GET /{cluster_id}/job-schedules`
Get the description of a scheduled job.	`GET /{cluster_id}/job-schedules/{schedule_id}`
Schedule a job.	`POST /{cluster_id}/job-schedules`
Modify a scheduled job.	`PUT /{cluster_id}/job-schedules/{schedule_id}`
Delete a scheduled job.	`DELETE /{cluster_id}/job-schedules/{schedule_id}`
Managing Snapshots	
List snapshots in the cluster.	`GET /{cluster_id}/backups`
List snapshots for a keyspace in the cluster.	`GET /{cluster_id}/backups/{ks_name}`
Run a snapshot immediately.	`POST /{cluster_id}/backups/run`
Restoring from Snapshots	
Restore all data from a snapshot.	`POST /{cluster_id}/backups/restore/{tag}`
Restore a specific keyspace from a snapshot.	`POST /{cluster_id}/backups/restore/{tag}/{ksname}`
Restore a specific column family from a snapshot.	`POST /{cluster_id}/backups/restore/{tag}/{ksname}`

Managing Snapshot Job Schedules

Job Schedule
A job schedule describes an action that OpsCenter will run periodically. Currently, the only type of scheduled job available is a snapshot job.
A job schedule has the following form:

```
{
    "first_run_date": FIRST_RUN_DATE,
    "first_run_time": FIRST_RUN_TIME,
    "timezone": TIMEZONE,
    "interval": INTERVAL,
    "interval_unit": INTERVAL_UNIT
    "job_params": JOB_PARAMS,
    "id": ID,
    "next_run": NEXT_RUN,
```

```
    "last_run": LAST_RUN
}
```

Data:

- **FIRST_RUN_DATE** (*string*) -- A date in YYYY-MM-DD format the date for running the job.
- **FIRST_RUN_TIME** (*string*) -- A time in hh:mm:ss format specifying the time to begin running the job.
- **TIMEZONE** (*string*) -- The time zone, listed in the OpsCenter /meta/timezones directory, for the job schedule. For example, GMT, US/Central, US/Pacific, and US/Eastern are valid timezones.
- **INTERVAL** (*int*) -- In conjunction with interval_unit, this controls how often the job is executed. For example, an interval of 2 and an interval_unit of weeks results in a job that will run every two weeks.
- **INTERVAL_UNIT** (*string*) -- The unit of time for interval. Values are minutes, hours, days, or weeks.
- **JOB_PARAMS** (*dict*) -- A dictionary that describes the job. Only the type field is required to be present in the dictionary, and its value is currently limited to backup. Fields for this dictionary are specific to the type; for the backup type, see **Backup Job Params**.
- **ID** (*string*) -- A unique ID that references a job schedule. Use only for retrieving a job schedule.
- **NEXT_RUN** (*string*) -- The date and time of the next scheduled run.
- **LAST_RUN** (*string*) -- The date and time of the last successful run.

Backup Job Params

A JSON dictionary that describes a backup type of **Job Schedule**. This should be used as the job_params field in the **Job Schedule**.

```
{
    "type": "backup",
    "keyspaces": KEYSPACES,
    "cleanup_age": CLEANUP_AGE,
    "cleanup_age_unit": CLEANUP_AGE_UNIT,
    "pre_snapshot_script": PRE_SNAPSHOT_SCRIPT,
    "post_snapshot_script": POST_SNAPSHOT_SCRIPT
}
```

Data:

- **KEYSPACES** (*list*) -- A JSON list of keyspace names that should be included in scheduled snapshots. An empty list or null will result in all keyspaces being included in the snapshots.
- **CLEANUP_AGE** (*int*) -- (Optional) In combination with CLEANUP_AGE_UNIT, this specifies the age at which old snapshots should be deleted. A value of 0 will disable automatic snapshot cleanup. This option defaults to a value of 0, meaning automatic snapshot cleanups are disabled by default.
- **CLEANUP_AGE_UNIT** (*string*) -- (Optional) The unit of time for CLEANUP_AGE. Valid values include "minutes", "hours", "days" (the default), and "weeks".
- **PRE_SNAPSHOT_SCRIPT** (*string*) -- (Optional) The file name of a custom script to be automatically run prior to triggering each snapshot. This file must exist within the bin/backup-scripts/ directory where the OpsCenter agent is installed. Only letters, numbers, underscores and hyphens are permitted in the name.
- **POST_SNAPSHOT_SCRIPT** (*string*) -- (Optional) The file name of a custom script to be automatically run after each snapshot is taken. The name of each file included in the snapshot will be passed to the script through stdin. This file must exist within the bin/backup-scripts/ directory where the OpsCenter agent is installed. Only letters, numbers, underscores and hyphens are permitted in the name.

GET/cluster_id/job-schedules

Retrieve a list of jobs scheduled to run in OpsCenter. Currently the only type of job is a scheduled snapshot.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Returns a list of **Job Schedule** objects.

Example:

```
curl http://127.0.0.1:8888/Test_Cluster/job-schedules
```

Output:

```
[
  {
    "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": "2012-04-20 18:00:00 GMT",
    "next_run": "2012-04-21 18:00:00 GMT",
    "timezone": "GMT"
  },
  ...
]
```

GET/cluster_id/job-schedules/schedule_id

Get the description of a scheduled job.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **schedule_id** -- A unique ID of the scheduled job that matches the id of a **Job Schedule** object.

Returns a **Job Schedule** object.

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 of -1 and interval_unit of null.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Body : A dictionary in the format of a **Job Schedule** describing the scheduled job to create. The id, last_run, and next_run fields should be omitted.

Responses: **201** -- Job schedule created successfully

Returns the ID of the newly created job.

Example

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/job-schedules/
  -d
  '{
    "first_run_date": "2012-05-03",
    "first_run_time": "18:00:00",
    "interval": 1,
    "interval_unit": "days",
    "job_params": {
        "cleanup_age": 30,
        "cleanup_age_unit": "days",
        "keyspaces": [],
        "type": "backup"
    },
    "timezone": "GMT"
  }'
```

Output:

```
"905391b7-1920-486d-a633-282f22dce604"
```

PUT/cluster_id/job-schedules/schedule_id

Update a scheduled job.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **schedule_id** -- A unique ID identifying the schedule job to update.

Body : A dictionary with fields from **Job Schedule** that you would like to update.

Responses: **200** -- Schedule updated successfully

Returns null.

Example

```
curl -X PUT
  http://127.0.0.1:8888/Test_Cluster/job-schedules/905391b7-1920-486d-a633-282f22dce604
  -d
  '{
    "interval": "12",
```

```
    "interval_unit": "hours"
  }'
```

DELETE/cluster_id/job-schedules/schedule_id

Delete a scheduled job.

Path arguments:
- **cluster_id** -- A **Cluster Config** ID.
- **schedule_id** -- A unique ID identifying the schedule job to delete.

Responses: **200** -- Schedule deleted successfully

Returns null.

Example

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

Managing Snapshots

GET/cluster_id/backups

Get a list of snapshots that exist in the cluster.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Opt. params: **match** -- A **Job Schedule** ID to limit snapshots to those created by a particular snapshot schedule.

Returns a dictionary describing 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
```

Output:

```
{
  "opscenter_adhoc_2012-04-19-17-41-12-UTC": {
    "id": "adhoc",
      "keyspaces": {
          "OpsCenter": {
                "cfs": [
                    "events",
                    "events_timeline",
                    "rollups60",
                    ...
                ],
                "nodes": [
                    "127.0.0.1"
                ],
                "size": 23180
            },
            "system": {
                "cfs": [
                    "LocationInfo",
                    "Schema",
                    ...
                ],
```

```
                "nodes": [
                    "127.0.0.1"
                ],
                "size": 48961
            },
            "test": {
                "cfs": [
                    "cf1",
                    "cf2"
                ],
                "nodes": [
                    "127.0.0.1"
                ],
                "size": 4543
            }
        },
        "time": 1334857272
    }
}
```

GET/cluster_id/backups/ks_name

Get a list of snapshots that exist for a keyspace in the cluster.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **ks_name** -- The keyspace scheduled for snapshot.

Returns data in the same format as `GET /{cluster_id}/backups`.

POST/cluster_id/backups/run

Run a one time backup immediately.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

Body : A JSON dictionary containing the options to use for running this one-time snapshot.

Returns true to indicate a successful snapshot.

Example

Start a snapshot of two keyspaces immediately.

```
curl -X POST
  http://127.0.0.1:8888/Test_Cluster/backups/run
  -d '{"keyspaces": ["Keyspace1", "Keyspace2"]}'
```

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 snapshot in the future using the `POST /{cluster_id}/job-schedules`. Specify an interval=-1 and interval_unit=null.

Restoring from Snapshots

POST/cluster_id/backups/restore/tag

Restore all data from a snapshot.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **tag** -- The ID of a snapshot to restore. This must match one of the IDs returned by `GET /{cluster_id}/backups`.

Body : A JSON dictionary of options, including:

- `throttle`: The streaming limit in MB/second. The default behavior is no throttle.
- `excluded-nodes`: The list of nodes to exclude from the restore process.
- `force`: True or false. True ignores problems, such as some nodes or agents being down, that normally abort restoration.
- `keyspaces`: A dictionary mapping keyspace names to a dictionary of options specific to that keyspace. If this parameter is omitted, all keyspaces will be restored. If it is present, only the keyspaces listed in the dictionary will be restored. The set of accepted options per-keyspace include the following:
  - `column-families`: A list of specific column families to restore. If omitted, all column families will be restored. If omitted, all column families will be restored.
  - `truncate`: True or false. True first truncates all column families that will be restored. By default, column families are not truncated before restoration.

Returns a **Request** ID.

Example

```
SNAPSHOT='opscenter_4a269167-96c1-40c7-84b7-b070c6bcd0cd_2012-06-07-18-00-00-UTC'
curl -X POST
    http://192.168.1.1:8888/Test_Cluster/backups/restore/$SNAPSHOT
    -d '{
      "throttle": 10,
      "excluded-nodes": [
        "192.168.100.7"
      ],
      "keyspaces": {
        "Keyspace1": {
          "column-families: ["users", "dates"],
          "truncate": true
        },
        "OpsCenter": {
          "truncate": false
        }
      },
    }'
```

Output:

```
"fee2232c-48ac-11e2-b1b9-e0b9a54a6d93"
```

POST/cluster_id/backups/restore/tag/ksname

Restore a keyspace from a snapshot.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **tag** -- The ID of a snapshot to restore. This must match one of the ID's returned by `GET /{cluster_id}/backups`.
- **ksname** -- A keyspace to restore.

Body : A JSON dictionary of options, including:

- `throttle`: The streaming limit in MB/second. The default behavior is no throttle.
- `excluded-nodes`: The list of nodes to exclude from the restore process.
- `force`: True or false. True ignores problems, such as some nodes or agents being down, that normally abort restoration.
- `column-families`: A list of specific column families to restore. If omitted, all column families will be restored.
- `truncate`: True or false. True first truncates all column families that will be restored. By default, column families are not truncated before restoration.

Returns a **Request** ID.

Example

```
SNAPSHOT='opscenter_4a269167-96c1-40c7-84b7-b070c6bcd0cd_2012-06-07-18-00-00-UTC'
curl -X POST
    http://192.168.1.1:8888/Test_Cluster/backups/restore/$SNAPSHOT/Keyspace1
    -d '{
      "throttle": 10,
      "excluded-nodes": [
        "192.168.100.7"
      ],
      "column-families: ["users", "dates"],
      "truncate": true
    }'
```

Output:

```
"fee2232c-48ac-11e2-b1b9-e0b9a54a6d93"
```

POST/cluster_id/backups/restore/tag/ksname/cfname

Restore a column family from a snapshot.

Path arguments:

- **cluster_id** -- A **Cluster Config** ID.
- **tag** -- The ID of a snapshot to restore. This must match one of the ID's returned by `GET /{cluster_id}/backups`.
- **ksname** -- A keyspace to restore.
- **cfname** -- A column family to restore.

Body : A JSON dictionary of options, including:

- `throttle`: The streaming limit in MB/second. The default behavior is no throttle.
- `excluded-nodes`: The list of nodes to exclude from the restore process.
- `force`: True or false. True ignores problems, such as some nodes or agents being down, that normally abort restoration.
- `truncate`: True or false. True first truncates all column families that will be restored. By default, column families are not truncated before restoration.

Returns a **Request** ID.

Example

```
SNAPSHOT='opscenter_4a269167-96c1-40c7-84b7-b070c6bcd0cd_2012-06-07-18-00-00-UTC'
curl -X POST
    http://192.168.1.1:8888/Test_Cluster/backups/restore/$SNAPSHOT/Keyspace1/users
    -d '{
      "throttle": 10,
      "excluded-nodes": [
        "192.168.100.7"
      ],
      "truncate": true
    }'
```

Output:

```
"fee2232c-48ac-11e2-b1b9-e0b9a54a6d93"
```

Hadoop Status

Hadoop Status Methods	URL
Get the current status of the Hadoop cluster.	`GET /{cluster_id}/hadoop/status`
Get a list of submitted jobs and their status.	`GET /{cluster_id}/hadoop/jobs`

GET/cluster_id/hadoop/status

Retrieve information about the status of the Hadoop cluster. Returns a JSON dictionary of cluster attributes and their current value.

Example:

```
curl http://127.0.0.1:8888/Test_Cluster/hadoop/status
```

Output:

```
{
    "activeTrackerNames": [
        "tracker_dse1:dse1/127.0.0.1:47333"
    ],
    "blacklisted_tracker_names": [],
    "has_recovered": false,
    "has_restarted": false,
    "hostname": "dse1",
    "http_port": 50030,
    "identifier": "201301301843",
    "mapTasks": 0,
    "max_map_tasks": 2,
    "max_memory": 4125097984,
```

```
    "max_reduce_tasks": 2,
    "num_active_trackers": 1,
    "num_blacklisted_trackers": 0,
    "num_excluded_nodes": 0,
    "reduce_tasks": 0,
    "start_time": 1359592991647,
    "state": 1,
    "task_tracker_expiry_interval": 600000,
    "total_submissions": 5,
    "used_memory": 4125097984
}
```


Job Details

GET/cluster_id/hadoop/jobs

Returns a JSON list of submitted jobs, each of which is a dictionary of job attributes.

Example:

```
curl http://127.0.0.1:8888/Test_Cluster/hadoop/jobs
```

Output:


```
[
    {
        "desired_maps": 1,
        "desired_reduces": 1,
        "duration": 18951,
        "finish_time": 0,
        "finished_maps": 1,
        "finished_reduces": 0,
        "job_id": {
            "as_string": "job_201301301843_0001",
            "job_id": 1,
            "job_tracker_id": "201301301843"
        },
        "launch_time": 1359593329324,
        "priority": 2,
        "profile": {
            "job_file": "cfs:/tmp/hadoop-joe/mapred/staging/joe/.staging/job_201301301843_0001/job.xml",
            "name": "INSERT OVERWRITE TABLE 10da...b.column_name)(Stage-1)",
            "queue_name": "default",
            "user": "joe"
        },
        "start_time": 1359593329128,
        "status": {
            "cleanup_progress": 0.0,
            "map_progress": 1.0,
            "reduce_progress": 0.0,
            "run_state": 1,
            "scheduling_info": "NA",
            "setup_progress": 1.0
        }
    }
]
```


Provisioning New Nodes and Clusters

Using these methods, you can create new clusters or add nodes to an existing cluster in an automated fashion. The nodes may be created on existing machines and VMs, or OpsCenter can launch EC2 instances to host the nodes.

Provisioning Methods	URL
Provision a new cluster	`POST /provision`
Add nodes to an existing cluster	`POST /{cluster_id}/provision`
Launch EC2 instances and provision a new cluster	`POST /launch`
Launch EC2 instances to add nodes to a cluster	`POST /{cluster_id}/launch`

Provisioning on Prepared Servers

POST/provision

Set up and start a new Cassandra or DSE cluster on existing machines.
The post body should be a JSON dictionary containing, at a minimum, the following required entries:

- nodes: A list of IP addresses to set up as part of the cluster
- cassandra_config: A **Cassandra Config**, which is a JSON dictionary representing the cassandra.yaml configuration for the nodes in the cluster.
- install_params: A dictionary specifying what packages to install and the user credentials needed to access the machines using SSH. The following entries are accepted:
  - package: Either "dsc" or "dse"
  - version: The version of DSC or DSE to install
  - username: The username to use when SSHing to the machines
  - password: The password for username. This may be omitted if a private key is specified and the user has root privileges or can perform the necessary actions using sudo without specifying a password.
  - private_key: The text contents of the private key for username. This may be omitted if a password is specified.
  - private_key_file: The path to a private key file. This may be used in place of the private_key option.
  - repo-user: If installing DSE, this is the username that you use to access the DataStax enterprise repositories.
  - repo-password: If installing DSE, this is the password that you use to access the DataStax enterprise repositories.

The following entries are typically optional, but are sometimes required:

- opscenter_config: A JSON dictionary representing the OpsCenter **Cluster Config** for the new cluster. If this option is omitted, the `[cassandra]: seed_hosts` option will be automatically populated and all other options will be left at their default values.
- node_type_map: The role for each node that you specify when provisioning a DSE cluster. The entry should be a map from IP addresses to one of the following: "cassandra", "hadoop", or "solr". If omitted, "cassandra" is assumed.
- topology_map: Specifications of the DCs and racks for the provisioned nodes, which you can pass as a map in the form `{ip: [dc, rack]}`. This argument is required when using a snitch other than SimpleSnitch, RackInferringSnitch, DseSimpleSnitch, or DseDelegateSnitch. The topology info is used for token selection purposes, but will *not* result in cassandra-topology.properties being filled out automatically.
- token_map: A map of the form `{ip: token}` that overrides the default OpsCenter selection of balanced tokens for each node on a per-DC basis. OpsCenter does not currently pick balanced tokens automatically when multiple racks are in use. This should only be supplied when provisioning a cluster that is not using vnodes.
- ssh_ip_map: A map of private interfaces to public interfaces used to deal with nodes that have different public and private interfaces (such as Amazon EC2).

- accepted_fingerprints: A parameter that SSH automatically checks for each host. If the fingerprint for a host has not been seen before, the operation will fail with a status code of 409 unless that fingerprint is included in the accepted_fingerprints parameter. The format should be a map like `{ip: <fingerprint>}`, where `<fingerprint>` should be the the result of:

  ```
  ssh-keygen -lf /dev/stdin <<< "_ $(ssh-keyscan $HOST 2>/dev/null | cut -f2,3 -d ' ')"
  ```

  Alternatively, when a 409 status is returned, the result body will be a JSON dictionary with a fingerprints entry. The value of that entry is a map of `{ip: <fingerprint>}` that can be directly used as the value for the accepted_fingerprints parameter.

Returns a **Request** ID.

Example

When launching a three node DSC cluster, Our post body might look like:

```
{
    "cassandra_config": {
        "authenticator": "org.apache.cassandra.auth.AllowAllAuthenticator",
        "authority": "org.apache.cassandra.auth.AllowAllAuthority",
        "auto_snapshot": true,
        "cluster_name": "Test Cluster",
        "column_index_size_in_kb": 64,
        "commitlog_directory": "/var/lib/cassandra/commitlog",
        "commitlog_sync": "periodic",
        "commitlog_sync_period_in_ms": 10000,
        "compaction_preheat_key_cache": true,
        "compaction_throughput_mb_per_sec": 16,
        "concurrent_reads": 32,
        "concurrent_writes": 32,
        "data_file_directories": [
            "/var/lib/cassandra/data"
        ],
        "dynamic_snitch_badness_threshold": 0.1,
        "dynamic_snitch_reset_interval_in_ms": 600000,
        "dynamic_snitch_update_interval_in_ms": 100,
        "encryption_options": {
            "internode_encryption": "none",
            "keystore": "conf/.keystore",
            "keystore_password": "cassandra",
            "truststore": "conf/.truststore",
            "truststore_password": "cassandra"
        },
        "endpoint_snitch": "SimpleSnitch",
        "flush_largest_memtables_at": 0.75,
        "hinted_handoff_enabled": true,
        "hinted_handoff_throttle_delay_in_ms": 1,
        "in_memory_compaction_limit_in_mb": 64,
        "incremental_backups": false,
        "index_interval": 128,
        "initial_token": null,
        "key_cache_save_period": 14400,
        "key_cache_size_in_mb": null,
        "max_hint_window_in_ms": 3600000,
        "memtable_flush_queue_size": 4,
        "multithreaded_compaction": false,
        "partitioner": "org.apache.cassandra.dht.RandomPartitioner",
        "reduce_cache_capacity_to": 0.6,
        "reduce_cache_sizes_at": 0.85,
```

```
            "request_scheduler": "org.apache.cassandra.scheduler.NoScheduler",
            "row_cache_provider": "SerializingCacheProvider",
            "row_cache_save_period": 0,
            "row_cache_size_in_mb": 0,
            "rpc_keepalive": true,
            "rpc_port": 9160,
            "rpc_server_type": "sync",
            "rpc_timeout_in_ms": 10000,
            "saved_caches_directory": "/var/lib/cassandra/saved_caches",
            "snapshot_before_compaction": false,
            "ssl_storage_port": 7001,
            "storage_port": 7000,
            "thrift_framed_transport_size_in_mb": 15,
            "thrift_max_message_length_in_mb": 16,
            "trickle_fsync": false,
            "trickle_fsync_interval_in_kb": 10240
        },
        "install_params": {
            "username": "joe",
            "password": "somepassword",
            "package": "dsc",
            "version": "1.1.2"
        },
        "nodes": [
            "192.168.100.1",
            "192.168.100.2",
            "192.168.100.3"
        ]
    }
```

Which we can use as follows:

```
curl -X POST
  localhost:8888/provision
  -d @provision.json
```

Output:

```
"da0794da-4a3a-11e2-b745-e0b9a54a6d93"
```

DSE Example
When launching a DSE cluster with two Cassandra nodes, one Hadoop node, and one Solr node, our post body might look like:

```
{
    "cassandra_config": {
        "authenticator": "org.apache.cassandra.auth.AllowAllAuthenticator",
        "authority": "org.apache.cassandra.auth.AllowAllAuthority",
        "auto_snapshot": true,
        ...
        "trickle_fsync_interval_in_kb": 10240
    },
    "install_params": {
        "username": "joe",
        "password": "somepassword",
        "package": "dse",
        "version": "2.2.1",
```

```
        "repo-user": "some-dse-username",
        "repo-password": "some-dse-password"
    },
    "nodes": [
        "192.168.100.1",
        "192.168.100.2",
        "192.168.100.3",
        "192.168.100.4"
    ],
    "node_type_map": {
        "192.168.100.1": "cassandra",
        "192.168.100.2": "cassandra",
        "192.168.100.3": "hadoop",
        "192.168.100.4": "solr"
    }
}
```

POST/cluster_id/provision

Add new Cassandra or DSE nodes to a cluster.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

The post body should be the same as for `POST /provision`, but with the following differences:

- opscenter_config is not accepted.
- token_map is required when provisioning a non vnode cluster.

Returns a **Request** ID.

Example

When adding two nodes to a DSC cluster, the post body may look like:

```
{
    "cassandra_config": {
        "authenticator": "org.apache.cassandra.auth.AllowAllAuthenticator",
        "authority": "org.apache.cassandra.auth.AllowAllAuthority",
        "auto_snapshot": true,
        ...
        "trickle_fsync_interval_in_kb": 10240
    },
    "install_params": {
        "username": "joe",
        "password": "somepassword",
        "package": "dsc",
        "version": "1.1.2"
    },
    "nodes": [
        "192.168.100.4",
        "192.168.100.5"
    ],
    "token_map": {
        "192.168.100.4": "10",
        "192.168.100.5": "46248042897083394072353090607538141429"
    }
}
```

Provisioning with New EC2 Instances

POST/launch

Launch a set of new EC2 instances and provision a new cluster on them.
The post body should be a JSON dictionary with two entries: launch and provision.
"launch" dictionary
The launch entry should be a dictionary containing the following required entries:

- ec2_access_id: The AWS Access ID for the account you wish to launch the instances with.
- ec2_secret_key: The matching AWS Secret Key for the provided Access ID.
- location: The AWS region to use for launching the instances. The following options are available:
  - "US East (Northern Virginia)"
  - "US West (Northern California)"
  - "US West (Oregon)"
  - "EU (Ireland)"
  - "Asia Pacific (Tokyo)"
  - "Asia Pacific (Singapore)"
  - "South America (Sao Paulo)"

  The default location is "US East (Northern Virginia)".

The following optional entries may also be used inside the launch dictionary:

- zone: The availability zone to launch the instances in. For example, options for the US East region include "us-east-1a", "us-east-1b", "us-east-1c", "us-east-1d", and "us-east-1e". If omitted, an availability zone will be randomly chosen.
- image_id: The AMI ID to use for the new instances. A good default choice for this is the DataStax AMI for the region. To see all options organized by region, look at `<opscenterd_conf_dir>/definitions/ec2-instances-*.json>`. By default, the DataStax AMI for US East will be used.
- image_size: The type and size of instances to launch. Options include “m1.small”, “m1.large”, “c1.medium”, and so on. Some sizes are only available in select AWS regions. By default, m1.large will be used.
- keypair: The name of an AWS keypair to use when launching the new EC2 instances. If omitted, a new key pair will be created with the name OpsCenterProvisioningKeyPair.
- security_group: The name of an AWS keypair to use when launching the new EC2 instances. If omitted, a new security group named OpsCenterSecurityGroup will be created and used. This security group opens the default ports used by Cassandra, DSE, OpsCenter, and SSH.

"provision" dictionary

- cassandra_config: A **Cassandra Config**, which is a JSON dictionary representing the cassandra.yaml configuration for the nodes in the cluster.

- install_params: A dictionary specifying what packages to install and the user credentials needed to access the machines using SSH. The following entries are accepted:
  - package: Either "dsc" or "dse"
  - version: The version of DSC or DSE to install
  - username: The username to use when SSHing to the machines
  - private_key: The text contents of the private key for username. This may be omitted if a password is specified or if the keypair option was omitted, which results in a new key pair being created.
  - private_key_file: The path to a private key file. This may be used in place of the private_key option.
  - repo-user: If installing DSE, this is the username that you use to access the DataStax enterprise repositories.
  - repo-password: If installing DSE, this is the password that you use to access the DataStax enterprise repositories.

The following entries are typically optional, but are sometimes required:

- opscenter_config: A JSON dictionary representing the OpsCenter **Cluster Config** for the new cluster. If this option is omitted, the `[cassandra]: seed_hosts` option will be automatically populated and all other options will be left at their default values.
- node_type_counts: When provisioning a DSE cluster, this allows you to specify how many nodes of each role type should be provisioned. It should be a map of the form `{node_type: count}`, where node_type is one of "cassandra", "hadoop", or "solr". For example, if `{"cassandra": 4, "hadoop": 2}` were used, the cluster would contain six nodes, with four running just DSE and two running hadoop services as well. When launching DSC clusters, only the `cassandra` type should be used.
- type_token_map: Normally, OpsCenter will select balanced tokens for each node on a per-DC basis. You can optionally override those selctions by passing in a map of the form `{node_type: [token, token, ...]}` where node_type is one of "cassandra", "hadoop", or "solr". This param should only be used when provisioning non vnode clusters.

Returns a **Request** ID.

Example

To launch a three node cluster in US East, availability zone 'a', our post body might look like this:

```
{
    "provision": {
        "cassandra_config": {
            "authenticator": "org.apache.cassandra.auth.AllowAllAuthenticator",
            "authority": "org.apache.cassandra.auth.AllowAllAuthority",
            "auto_snapshot": true,
            ...
            "trickle_fsync_interval_in_kb": 10240
        },
        "install_params": {
            "package": "dsc",
            "version": "1.1.2",
            "username": "ubuntu"
        },
        "node_type_counts": {
            "cassandra": 3
        }
    },

    "launch" {
        "ec2_access_id": "9BATIQ711LYY326X6191",
        "ec2_secret_key": "I2nr9Jk1welj1IJekIejkbnQ91JmajIJaRea2ajR",
        "location": "US East (Northern Virginia)",
```

```
            "zone": "us-east-1a",
            "image_id": "ami-6139e708",
            "image_size": "m2.xlarge",
            "keypair": "MyKeyPair",
            "security_group": "default"
        }
    }
```

POST/cluster_id/launch

Launch a set of new EC2 instances in order to add new nodes to an Cassandra or DSE cluster.

Path arguments: **cluster_id** -- A **Cluster Config** ID.

The post body should be the same as the one used for `POST /launch` with the following differences in the provision dictionary:

- opscenter_config should be omitted.
- node_type_counts is required only when provisioning a vnode cluster
- type_token_map is required when provisioning a non vnode cluster (and replaces node_type_counts).

Returns a **Request** ID.

Example

To launch two EC2 instances and add two Hadoop nodes and one Solr node to an existing DSE cluster, body might look like this:

```
{
    "provision": {
        "cassandra_config": {
            "authenticator": "org.apache.cassandra.auth.AllowAllAuthenticator",
            "authority": "org.apache.cassandra.auth.AllowAllAuthority",
            "auto_snapshot": true,
            ...
            "trickle_fsync_interval_in_kb": 10240
        },
        "install_params": {
            "package": "dse",
            "version": "2.2.1",
            "username": "ubuntu",
            "repo-user": "some-dse-username",
            "repo-password": "some-dse-password"
        },
        "type_token_map": {
            "hadoop": ["10", "86248042897083394072353090607538141429"],
            "solr": ["46248042897083394072353090607538141429"]
        }
    },

    "launch" {
        "ec2_access_id": "9BATIQ711LYY326X6191",
        "ec2_secret_key": "I2nr9Jk1welj1IJekIejkbnQ91JmajIJaRea2ajR",
        "location": "US East (Northern Virginia)",
        "zone": "us-east-1a",
        "image_id": "ami-6139e708",
        "image_size": "m2.xlarge",
        "keypair": "MyKeyPair",
        "security_group": "default"
```

```
        }
    }
```

Troubleshooting

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

Internet Explorer browser is not supported

If you try to load the OpsCenter client in Microsoft Internet Explorer, a dialog displays informing you that it is not supported.

OpsCenter is only supported on the latest versions of:

- Mozilla Firefox
- Google Chrome
- Apple Safari

Provisioning

General Troubleshooting Steps

- Ensure firewalls are *properly opened* between the `opscenterd` machine and each node.
- Check the following files (on any of the nodes having problems) for any errors:
  - `/var/log/cassandra/system.log`
  - `/var/log/opscenter-agent/agent.log`
- Verify that Cassandra (or DSE) was not previously installed on any of the machines; if it was, all binaries, configuration files, logs, etc. must be cleared out first.

Invalid Repository Credentials

Debian

When installing DSE, if you enter invalid values for DataStax Credentials, an error dialog displays text along the lines of:

> Installation stage failed: Installation failed on node 172.16.1.2: 'apt-get update' failed.

Clicking **Details** displays the entire output from both `stdout` and `stderr`. If, in this output, you see the "401 Unauthorized", it means that the credentials entered were invalid.

RHEL

When installing DSE, if you enter invalid values for DataStax Credentials, an error dialog displays text along the lines of:

> Installation failed on node 172.16.1.2: 'yum install' failed.

Clicking **Details** displays the entire output from both `stdout` and `stderr`. If, in this output, you see "The requested URL returned error: 401", it means that the credentials used were invalid.

Timed out waiting for Cassandra to start

If you receive this error, it most likely means that the Cassandra process failed to start on one or more nodes. You should look in `/var/log/cassandra/system.log` for any errors that may need to be resolved.

The following packages are already installed

If you receive an error that starts with this message, it means Cassandra (or DSE) is already installed on the system. OpsCenter provisioning requires that any instances of Cassandra (or DSE) be completely removed or purged before provisioning a new cluster.

Agents cannot connect to opscenterd

If you receive an error message that includes "The installed agent doesn't seem to be responding", there is most likely a firewall issue preventing the installed agent from connecting to the `opscenterd` machine. You should ensure that port 61620 is open on the `opscenterd` machine and check the agent logs.

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`.

Python used to run OpsCenter not built with SSL

In order to protect your AWS credentials when launching EC2 instances, OpsCenter needs to use HTTPS, but if the version of Python that is running opscenterd was not compiled with SSL support OpsCenter will not run even if SSL has been disabled in the configuration file.

To resolve the issue, first ensure that OpenSSL 0.9.8 is installed on your system. If you have compiled Python manually, it is recommended that you install Python 2.6+ through your package manager. On CentOS and RedHat Enterprise Linux, this is most easily done through EPEL packages.

If you must compile Python manually, make sure that SSL support is enabled. This blog post explains the process for Python 2.5, but the same steps will work for Python 2.6 or 2.7.

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 3.1.1*
- *OpsCenter 3.1*
- *OpsCenter 3.0.2*
- *OpsCenter 3.0*
- *OpsCenter 2.1.3*
- *OpsCenter 2.1.2*
- *OpsCenter 2.1*
- *OpsCenter 2.0*

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

OpsCenter 3.1.1

Resolved issues

- Fixed an issue with opscenterd not starting up when certain types of CQL3 column families exist.

OpsCenter 3.1

Features

- Column families created via CQL3 are recognized and monitored.
- Added support for provisioning nodes with vnodes enabled.

Resolved issues

- Fixed OpsCenter not running without the proper SSL library being installed, even if SSL feature has been disabled, has been fixed.
- Fixed an issue with editing keyspace properties if another keyspace was recently updated.
- Fixed loading Performance presets with special characters in the name.
- Fixed agent hitting out of memory exception when cleaning up some backups.
- Fixed the List View rendering when nodes have long hostnames.
- Fixed an issue with adding nodes to an existing DSE cluster when some nodes were down.
- Fixed an issue when provisioning nodes using RackInferringSnitch and hostnames.

OpsCenter 3.0.2

OpsCenter Enterprise Edition 3.0.2 changes include the following bug fixes:

Resolved issues

- When adding internal Cassandra authentication to a cluster, the cluster's edited configuration can now be saved. Previoiusly an error was reported: Unable to connect to cluster.
- Scrolling long lists of column families in the Schema and Data Explorer panes has been implemented.
- Agents now receive configuration changes without having to be restarted.
- Compaction status now displays properly in List View or node popup.
- OpsCenter no longer becomes unresponsive (caused by an agent hanging) if there are too many files in the snapshots directory.

OpsCenter 3.0

OpsCenter Enterprise Edition 3.0 supports Cassandra 1.2 when virtual nodes are disabled.

What's new

- Visual cluster creation and management
    - Managing settings
    - Restarting a cluster
- Database restore
- Customizable backups
- Software update notifications
- Improved object creation and management
- Automated collection of diagnostic data
- Offloading of metadata repository to secondary clusters
- DSE security and encryption

Resolved issues

- Fixed node popup in Ring/Physical view going blank.
- Fixed agent data collection being interrupted on error.
- Fixed display of some data in Performance section.
- Fixed list of nodes in List View jumping around when compactions are running.
- Fixed several minor issues in the UI.
- Fixed several issues with installing opscenterd and the agents.

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

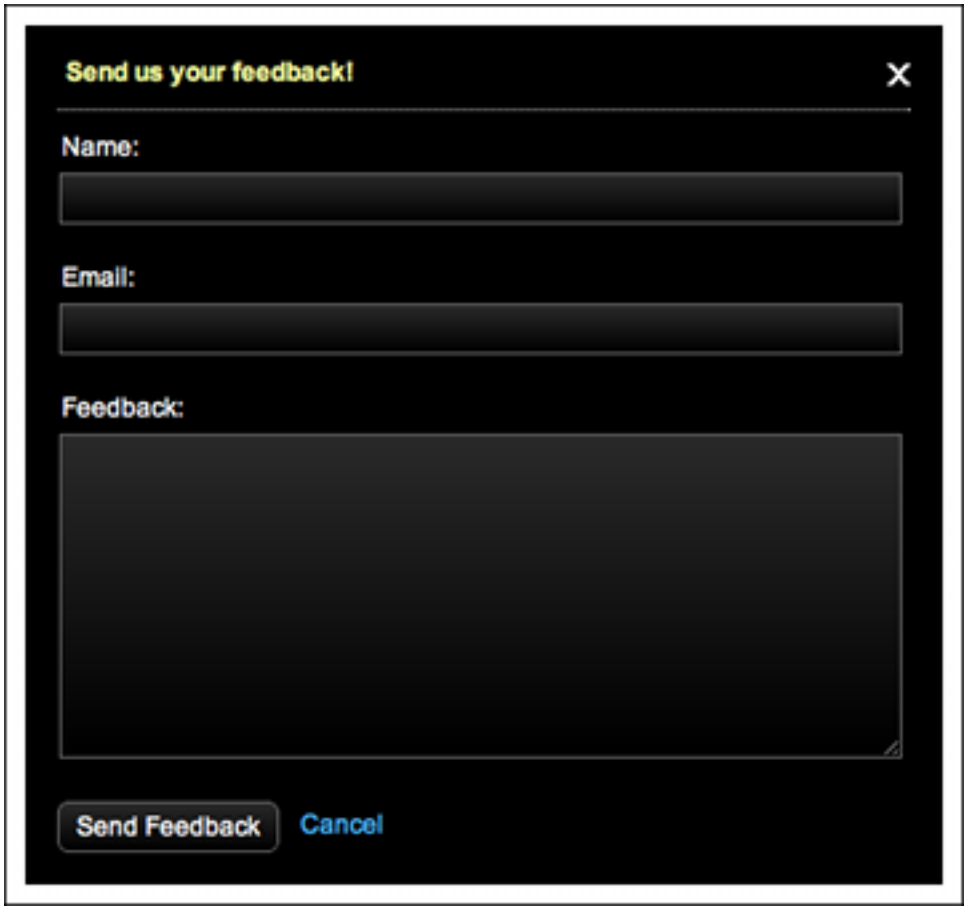
- Multi-cluster management. You can manage multiple cluster from the same OpsCenter instance.
- New Overview page for monitoring multiple clusters. It shows a condensed view of each cluster's Dashboard. It is visible only when multiple clusters exist.
- Perform and schedule operations from OpsCenter.
- 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

- Fixed 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.
- Fixed agent listen address when SSL is disabled.
- Fixed data explorer error when searching for row key containing "#".

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.