Pivotal

Table of Contents

Pivotal Greenplum Command Center Documentation

Documentation for Pivotal Greenplum Command Center.

- About Greenplum Command Center
  Pivotal Greenplum Command Center is a management tool for the Greenplum Big Data Platform. This section introduces key concepts about Greenplum Command Center and its components.
- Installing the Greenplum Command Center Software
  Downloading and installing the Greenplum Command Center software in your Greenplum cluster and one-time tasks. This section also provides instructions for upgrading Command Center to a new release and migrating Command Center instances.
- Creating Greenplum Command Center Console Instances
  Creating a Command Center Console instance to manage a Greenplum cluster.
- Using the Greenplum Command Center Web Interface
  Using the Greenplum Command Center web user interface to monitor and manage a Greenplum cluster.
- Administering Greenplum Command Center
  System administration information for the Greenplum Command Center components.
- Utility Reference
  Reference information for the two Greenplum Command Center utility programs: the `gpperfmon_install` utility that enables the data collection agents and the `gpcmdr` utility that sets up and manages the web application.
- Configuration File Reference
  References for Greenplum Command Center configuration files.
- Command Center Database Reference
  References for the Greenplum Command Center `gpperfmon` database tables.

About Pivotal Greenplum Command Center

Pivotal Greenplum Command Center is a management tool for the Greenplum Big Data Platform. This section introduces key concepts about Greenplum Command Center and its components.

Introduction

Greenplum Command Center monitors system performance metrics, analyzes system health, and allows administrators to perform some management tasks in a Greenplum environment. The Greenplum Command Center Console is an interactive graphical web application that is installed on a web server, usually on the master host. Users view and interact with the collected Greenplum system data through this application.

Greenplum Command Center is comprised of data collection agents that run on the master host and each segment host. The agents collect data about queries and system utilization and update the Greenplum master host at regular intervals. Greenplum Command Center stores its data and metrics in a dedicated Greenplum database (the Command Center database, gpperfmon) whose information is distributed among the master host and segment hosts like any other Greenplum Database. You can access the data stored in the Command Center database through the Greenplum Command Center Console and through SQL queries.

Note: Command Center requires Greenplum Database to operate because Command Center stores its information in a Greenplum database.

Supported Greenplum Platforms

Greenplum Command Center is currently certified for the Greenplum Data Computing Appliance (DCA) and Greenplum Database software-only environments. Command Center monitors the following for each environment:

Greenplum Data Computing Alliance:

- Greenplum Database Module

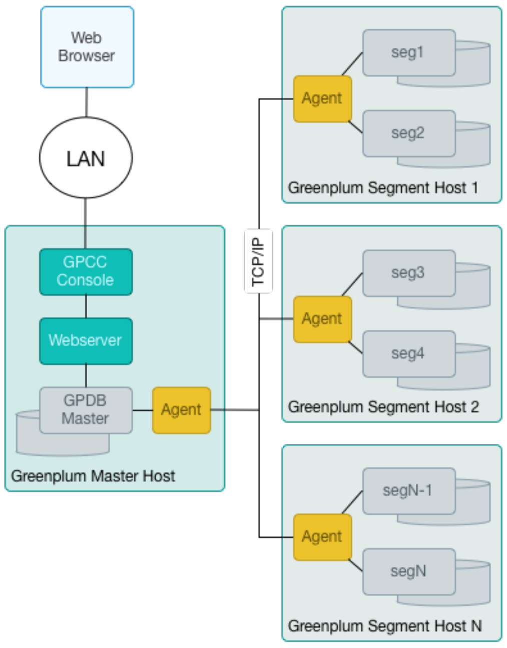
Greenplum Database (software-only environments):

- Greenplum Database

See the *Release Notes* for your Greenplum Command Center release for information about the supported software and hardware versions.

Architecture

The following figure illustrates the Greenplum Command Center architecture.



Greenplum Data Collection Agents

Greenplum data collection agents run on Greenplum segment hosts to collect the query and system statistics to be displayed in the Command Center. The agents are installed with the Greenplum Database distribution but are not enabled until you create the Greenplum Command Center database (gpperfmon) and start them. The agents may also be enabled or disabled by setting the `gp_enable_gpperfmon` server configuration parameter. When this parameter is enabled, the data collection agents run on all Greenplum hosts (master and segments), and start and stop along with Greenplum Database server processes.

The master agent polls all segment agents for system metrics and other data at a configurable interval (called the quantum). The master agent amasses the data from all segments, stores it in flat files, and periodically commits the data in the files to the Greenplum Command Center database.

Greenplum Command Center Database

The Greenplum Command Center database (gpperfmon) is a database within your Greenplum system dedicated to storing and serving system data. Your Greenplum Database installation includes the `gpperfmon_install` utility to install the Command Center database and optionally start the data collection agents.

When this document refers to the Command Center database, it is referring to the database named gpperfmon.

Greenplum administrators can connect to the Command Center database using client programs such as psql or application programming interfaces (APIs) such as JDBC (Java Database Connectivity) and ODBC (Open Database Connectivity). Administrators can also use the Greenplum Command Center Console to view reports on current and historical performance and perform other management tasks.

The Command Center database consists of three sets of tables; *now* tables store data on current system metrics such as active queries, *history* tables store data on historical metrics, and *tail* tables are for data in transition. Tail tables are for internal use only and should not be queried by users. The now and tail data are stored as text files on the master host file system, and the Command Center database accesses them via external tables. The history tables are regular database tables stored within the Command Center (gpperfmon) database. See Command Center Database Reference for the schema definitions of these tables.

Greenplum Command Center Console

Greenplum Command Center provides a browser-native HTML5 graphical console for viewing Greenplum Database system metrics and performing certain database administrative tasks. This browser-based application provides the following functionality:

- Interactive overview of realtime system metrics
- Detailed realtime statistics for the cluster and by server
- Query Monitor view lists queries executing and waiting to execute
- Four permission levels to allow users to view or cancel their own or others' queries, and to view or manage administrative information
- Cluster Metrics view shows synchronized charts of historical system metrics
- History view lists completed queries and system metrics plotted over a selected time period
    - Select a query to view query text and explain plan
- Segment Status view with summaries and details by segment
- Storage Status view with summaries and details by segment data directory
- Admin > Permissions view to see or manage permission levels
- Admin > Authentication view to see or edit the `pg_hba.con` host-based authentication configuration file

If you have multiple Greenplum environments, you can create separate Command Center instances for them. Each separate console instance operates on a unique port and has its own unique configuration options. For more information, see Creating Greenplum Command Center Console Instances.

Greenplum Command Center Web Service

The Greenplum Command Center Console queries the Command Center database through a web service framework composed of a lightweight go-based web server.

The console setup utility, `gpcmdr`, sets up the web server and web service, prompting you for basic configuration information on the desired port and SSL options. Under normal conditions, the web server and web service API require minimal maintenance and administration, as described in Web Server

Administration.

Installing the Greenplum Command Center Software

The Greenplum Workload Manager installer is included in the Greenplum Command Center installer. Install Workload Manager using its bundled installer. See the Greenplum Workload Manager documentation for instructions to run the Greenplum Workload Manager installer.

Installation Notes

Greenplum Command Center may be installed on a Dell EMC Data Computing Appliance (DCA) or in a Greenplum Database software environment.

Command Center is compatible with the Data Computing Appliance (DCA), versions 1.2.x, 2.x, and 3.x. Visit Pivotal Network to download an installer for the most recent 3.x version for your DCA. See the Dell EMC DCA documentation for information about installing the software on your appliance.

The Greenplum Command Center software is typically installed on the Greenplum Database master host. Installing on the master host provides the best performance and security, since the Command Center Console database requests are not passed over the network.

If you are installing the Command Center Console on a remote system (that is, not the same system on which you installed Greenplum Database), you must first install the Greenplum Database binary software files on the remote system. Note that you do not need to initialize the database on the remote system. See the *Greenplum Database Installation Guide*for help installing the Greenplum Database software.

If you want to use features of Greenplum Command Center 2.x that are not available in the 3.x release, see Running Greenplum Command Center 2.x in Parallel With 3.x.

This section contains the following topics:

Downloading and Running the Greenplum Command Center Installer

Download and Run the Installer

Important: The `gpadmin` user owns and executes the Greenplum Command Center software, which is installed in the `/usr/local` directory by default. Before you begin installing, ensure that the `gpadmin` user has write permission in the directory where you will install the software. Be sure to prepare the standby master host in the same way. Alternatively, you can run the installer as `root` and, after installation, change the owner of the installation directory and its contents to `gpadmin`.

Follow these steps as `gpadmin` to download and install the Greenplum Command Center software.

1. Download the Greenplum Command Center installer file from the Greenplum Database section of Pivotal Network. Installer files are available for Linux 64-bit platforms.

2. Unzip the installer file.

```
$ unzip greenplum-cc-web-3.X.X-LINUX-x86_64.zip
```

3. Launch the installer with `bash`:

```
$ /bin/bash greenplum-cc-web-3.X.X-LINUX.bin
```

4. Read through the license agreement. When you reach the bottom, type `yes` to accept the license agreement.

5. The installer prompts you to provide an installation path. Press **Enter** to accept the default installation path (`/usr/local/greenplum-cc-web-X.X.X`), or enter an absolute path to another install location. You must have write permission in the location you specify.

6. The installer then asks if you want to install on the standby master.
   - Enter `no` if you do not want to install the software to the standby master host now. You can install the software to the standby master later using the `gpccinstall` utility. See Install Greenplum Command Center Software on Additional Hosts.
   - Enter `yes` to install the Greenplum Command Center software on the standby master host, then enter the name of the host.

7. The installer completes with a summary of the actions that were performed.

Note:
If you have performed the previous steps as any user other than `gpadmin`, you need to change ownership and permissions of the installation directory before you continue.

Change the ownership of the installation directory:

```
# chown -R gpadmin:gpadmin greenplum-cc-web-X.X.X
```

Change the permissions of the installation directory:

```
# chmod -R 755 greenplum-cc-web-X.X.X
```

Install Greenplum Command Center Software on Additional Hosts

Follow the steps in this section to install the Greenplum Command Center software on the standby master host or on other hosts where you want to run Greenplum Command Center console instances.

Run the `gpccinstall` utility as the `gpadmin` user on the host where you installed the Greenplum Command Center software.

1. Create a text file containing the names of the standby master host and other hosts where the software is to be installed, one name per line. Do not include the name of the host where you ran the installer. Hostnames must be resolvable in DNS. For example:

```
smdw
gpspare
```

2. Source the Greenplum Database and Command Center path files.

   ```
   $ source /usr/local/greenplum-db/greenplum_path.sh
   $ source /usr/local/greenplum-cc-web/gpcc_path.sh
   ```

3. As `gpadmin`, run the `gpccinstall` utility to install Command Center on all hosts listed in the host file you created.

   ```
   $ gpccinstall -f hostfilename
   ```

   where `hostfilename` is the name of the host file you created.

About the Command Center Installation

The installation and setup procedures create a software installation directory and a directory containing files and folders to support each Greenplum Command Center Console instance.

Software Installation Directory

The following files and first-level subdirectories are copied into the installation folder that you specified when you installed Greenplum Command Center Console. This location is referred to as `$GPPERFMONHOME`.

- `gpcc_path.sh` – file containing environment variables for Command Center
- `bin` – program files for Greenplum Command Center
- `etc` – contains `openssl.cnf` file
- `gpcc-wlm-<version>-<platform>.bin` – installer for Greenplum Workload Manager
- `instances` – contains a subdirectory of resources for each Greenplum Database instance monitored by the console
- `lib` – library files for Greenplum Command Center
- `open_source_licenses_GPCC.txt` – licenses for open source components used by Greenplum Command Center
- `www` – web server and user interface files

Instances Directory

The `$GPPERFMONHOME/instances` directory contains subdirectories named for each instance created during console setup. The `conf` subdirectory contains configuration files that you can edit. Other files and folders are used by the web services for the instance, and should not be modified or deleted.

Each subdirectory contains the following files and first-level subdirectories:

- `conf` – multi-cluster configuration file, `clusters.conf`
- `webserver` – web server logs for this instance and symbolic links to web server files in the installation directory

Setting the Greenplum Command Center Environment

To enable the `gpadmin` user to execute Command Center utilities such as `gpcmdr`, follow these steps to set up the Greenplum Command Center environment.

1. Add the `GPPERFMONHOME` environment variable to your startup shell profile (such as `~/.bashrc`). Set the variable to the Greenplum Command Center home directory.

   ```
   GPPERFMONHOME=/usr/local/greenplum-cc-web
   source $GPPERFMONHOME/gpcc_path.sh
   ```

   Ensure that the `$GPPERFMONHOME/gpcc_path.sh` file has entries for the `greenplum_path.sh` file and the `MASTER_DATA_DIRECTORY` environment variable. See the *Greenplum Database Installation Guide*for details.

2. Save and source the `.bashrc` file:

   ```
   $ source ~/.bashrc
   ```

Creating the gpperfmon Database

This topic describes how to create the Command Center `gpperfmon` database and enable the data collection agents. This task must be completed one time for the Greenplum Database system, before you create a Greenplum Command Center instance.

When the data collection agents are enabled, their processes are started and stopped (using `gpstart` and `gpstop`) on the Greenplum segment hosts along with the Greenplum Database server processes.

Greenplum provides a `gpperfmon_install` utility that performs the following tasks:

- Creates the Command Center database (gpperfmon).
- Creates the Command Center superuser role (`gpmon`).
- Configures Greenplum Database server to accept connections from the `gpmon` role (edits the `pg_hba.conf` and `.pgpass` files).
- Sets the Command Center server configuration parameters in the Greenplum Database server `postgresql.conf` files.

The `gpperfmon_install` utility and the agents are part of the Greenplum Database software distribution. The tasks in this topic can be performed before or after the Command Center software is installed.

Enabling the Collection Agents

1. Log in to the Greenplum master host as the `gpadmin` user.

   ```
   $ su - gpadmin
   ```

2. Source the path file from the Greenplum Database installation directory:

   ```
   # source /usr/local/greenplum-db/greenplum_path.sh
   ```

3. Run the `gpperfmon_install` utility with the `--enable` option. You must supply the connection port of the Greenplum Database master server process, and set the password for the `gpmon` superuser that will be created. For example:

   ```
   $ gpperfmon_install --enable --password changeme --port 5432
   ```

   Note:
   The `gpperfmon_install` utility creates entries for the `gpmon` user in the `$MASTER_DATABASE/pg_hba.conf` file. See gpmon User Authentication for notes about restricting the gpmon user's access to databases.
   The password you specify is saved in a `.pgpass` file in the `gpadmin` user's home directory. See Changing the gpmon Password for steps to change the `gpmon` password.

4. When the utility completes, restart Greenplum Database server. The data collection agents will not start until the database is restarted.

   ```
   $ gpstop -r
   ```

5. Using the `ps` command, verify that the data collection process is running on the Greenplum master. For example:

   ```
   $ ps -ef | grep gpmmon
   ```

6. Run the following command to verify that the data collection processes are writing to the Command Center database. If all of the segment data collection agents are running, you should see one row per segment host.

   ```
   $ psql gpperfmon -c 'SELECT * FROM system_now;'
   ```

The data collection agents are now running, and your Greenplum system now has a gpperfmon database installed. This is the database where Command Center data is stored. You can connect to it as follows:

```
$ psql gpperfmon
```

Configuring a Standby Master Host (if enabled)

1. Copy the `$MASTER_DATA_DIRECTORY/pg_hba.conf` file from your primary master host to your standby master host. This ensures that the required connection options are also set on the standby master.

2. Copy your `~/.pgpass` file from your primary master host to your standby master host. This file usually resides in the `gpadmin` user's home directory. Note that the permissions on `.pgpass` must be set to 600 (for example: `chmod 0600 ~/.pgpass`).

gpmon User Authentication

The `gperfmon_install` utility adds the `gpmon` user to the `pg_hba.conf` authorization configuration file entries allowinglocal connections to any database in the Greenplum cluster. Greenplum Command Center requires that the `gpmon` user have access to `gpperfmon` and every database that Command Center will monitor. Since the `gpmon` role is a Greenplum Database superuser, you may wish to restrict the role from accessing other databases. Edit the `$MASTER_DATA_DIRECTORY/pg_hba.conf` and edit these lines:

```
local   gpperfmon  gpmon        md5
host    all        gpmon        127.0.0.1/28    md5
```

List `gpperfmon` and the databases you want to monitor with Command Center in the second field:

```
local   gpperfmon,userdb1,userdb2   gpmon                   md5
host    gpperfmon,userdb1,userdb2   gpmon       127.0.0.1/28    md5
```

See Changing the gpmon Password for steps to change the `gpmon` user's password.

Upgrading Greenplum Command Center

This section provides steps for upgrading Pivotal Greenplum Command Center to a new version.

Upgrading Greenplum Command Center requires installing the new distribution, and then migrating Command Center instances from a previous installation.

A new Greenplum Command Center software release may be installed in the same parent directory as the current release, by default `/usr/local`. The installer updates the symbolic link `greenplum-cc-web` to point to the new release directory and leaves the old release directory in place. After the software is installed, run the `gpcmdr --migrate` command to recreate your Command Center instances.

Install the New Software Release

1. Log in as the `gpadmin` user.
2. Source the `greenplum_path.sh` and `gpcc_path.sh` files from the current release:

   ```
   $ source /usr/local/greenplum-db/greenplum_path.sh
   $ source /usr/local/greenplum-cc-web/gpcc_path.sh
   ```

3. Download the latest Command Center release from Pivotal Network. Installer files are available for Linux 64-bit platforms, and have names in the format:

   ```
   greenplum-cc-web-X.X.X-PLATFORM.zip
   ```

4. Unzip the installer file. For example:

   ```
   # unzip greenplum-cc-web-X.X.X-PLATFORM.zip
   ```

5. Launch the installer for the new release with the bash shell:

   ```
   $ /bin/bash greenplum-cc-web-X.X.X-PLATFORM.bin
   ```

   Note: The installer requires write permission in the installation directory (`/usr/local`, by default). If the `gpadmin` user does not have write permission in the installation directory, run the installation as `root`. You will need to change file ownership and permissions after the software is installed.

6. Read through the license agreement. When you reach the bottom, type `yes` to accept the license agreement.
7. The installer prompts you to provide an installation path. Enter a full path or press **ENTER** to accept the default, `/usr/local`. You must have write permission in the directory you specify.
8. If you ran the installation as `root` or any user other than `gpadmin`, change the ownership and permissions of the installation directory:

   ```
   # chown -R gpadmin:gpadmin /usr/local/greenplum-cc-web-versionx.x
   # chmod -R 755 /usr/local/greenplum-cc-web-versionx.x
   ```

   Change to the `gpadmin` user before you continue to the next step:

   ```
   # su - gpadmin
   ```

9. Ensure that you have a current host file listing the names of all of the other hosts participating in the Greenplum Database cluster, including the standby master host. The host names must be resolvable in DNS.
10. As `gpadmin`, run the `gpccinstall` utility to install the new Command Center files on all hosts listed in the host file:

    ```
    $ gpccinstall -f hostfilename
    ```

    where `hostfilename` is the name of the host file you created.

Migrate Command Center Instances

After the new Command Center software is installed, migrate your instances by running the `gpcmdr --migrate` command.

To migrate all instances from a previous installation, run `gpcmdr --migrate` with no arguments. For example:

```
$ gpcmdr --migrate
```

To migrate a single instance, run `gpcmdr --migrate <instance_name>` and, when prompted, provide its full installation path. For example:

```
$ gpcmdr --migrate myinstance
```

Custom changes to the `ssh-wrapper` file are not handled by instance migration. See Configuration File Reference for information about the `ssh-wrapper` file. If you have set a custom `ssh` path in this file, you must copy it to the current installation.

For example:

```
cp /usr/local/greenplum-cc-web-3.0.0/bin/ssh-wrapper /usr/local/greenplum-cc-web/bin
```

See Creating Greenplum Command Center Console Instances for instructions to create new instances with `gpcmdr --setup`.

Uninstalling Greenplum Command Center

To uninstall Greenplum Command Center, you must stop both the Command Center Console and disable the data collection agents. Optionally, you may also remove any data associated with Greenplum Command Center by removing your Command Center Console installation and the Command Center database.

1. Stop Command Center Console if it is currently running. For example:

   ```
   $ gpcmdr --stop
   ```

2. Remove the Command Center installation directory from all hosts. For example:

   ```
   $ rm -rf /usr/local/greenplum-cc-web-version
   ```

3. Disable the Data Collection Agents.

   a. Log in to the master host as the Greenplum administrative user (`gpadmin`):

      ```
      $ su - gpadmin
      ```

   b. Edit the `$MASTER_DATA_DIRECTORY/postgresql.conf` file and disable the data collection agents:

      ```
      gp_enable_gpperfmon = off
      ```

   c. Remove or comment out the gpmon entries in `pg_hba.conf`. For example:

      ```
      #local    gpperfmon    gpmon    md5
      #host     gpperfmon    gpmon    0.0.0.0/0    md5
      ```

   d. Drop the Command Center superuser role from the database. For example:

      ```
      $ psql template1 -c 'DROP ROLE gpmon;'
      ```

   e. Restart the Greenplum Database instance:

      ```
      $ gpstop -r
      ```

   f. Clean up any uncommitted Command Center data and log files that reside on the master file system:

      ```
      $ rm -rf $MASTER_DATA_DIRECTORY/gpperfmon/data/*
      $ rm -rf $MASTER_DATA_DIRECTORY/gpperfmon/logs/*
      ```

   g. If you do not want to keep your historical Command Center data, drop the gpperfmon database:

      ```
      $ dropdb gpperfmon
      ```

Running Greenplum Command Center 2.x in Parallel With 3.x

To use features of GPCC 2.x not available in 3.x, you can run a GPCC 2.x instance on a separate port on the master host and manage it independently from GPCC 3.x.

Continue Running an Installed GPCC 2.x Version

After you install GPCC 3.x:

1. Run `gpcmdr --status` to identify an available port that is not in use by a GPCC 3.x instance.
2. Change the symbolic link in your installation directory to the 2.x version:

   ```
   $ ln -sfn /usr/local/greenplum-cc-web-2.x.x /usr/local/greenplum-cc-web
   ```

3. Edit `$GPPERFMONHOME/instances/<instance_name>/conf/lighttpd.conf`.
   - Change the `server.port` parameter so it does not conflict with the 3.x port. For example:

   ```
   server.port = 28090
   ```

   - Change the seven occurrences of `greenplum-cc-web` to the absolute path of the 2.x installation directory. For example:

   ```
   server.document-root = "/usr/local/greenplum-cc-web/./instances/demo/web"
   ```

   becomes

   ```
   server.document-root = "/usr/local/greenplum-cc-web-2.5.0/instances/demo/web"
   ```

4. Start the GPCC 2.x instance:

   ```
   $ gpcmdr --start <instance_name>
   ```

5. Change the symbolic link in your installation directory to point to your current 3.x version:

   ```
   $ ln -sfn /usr/local/greenplum-cc-web-3.0.0 /usr/local/greenplum-cc-web
   ```

Install GPCC 2.x in Addition to GPCC 3.x

After you install GPCC 3.x:

1. Run `gpcmdr --status` to identify an available port not in use by a GPCC 3.x instance.
2. Download the GPCC 2.5.0 installer from Pivotal Network.
3. Unzip and run the installer:

   ```
   $ unzip greenplum-cc-web-2.5.0-RHEL5-x86_64.zip
   $ bash greenplum-ccweb-2.5.0-RHEL5-x86_64.bin
   ```

   This will change the symlink at `/usr/local/greenplum-cc-web` to the GPCC 2.5.0 installation directory.
   Note: Do not run `gpccinstall`, just the downloaded installer binary.

4. Create a GPCC 2.x instance:

   ```
   $ gpcmdr --setup
   ```

   Specify a port that is not used by any existing GPCC 3.x instances.

5. Edit `$GPPERFMONHOME/instances/<instance_name>/conf/lighttpd.conf`.

- Change the seven references to the `greenplum-cc-web` symbolic link to the absolute path of the GPCC 2.x installation directory. For example:

```
server.document-root = "/usr/local/greenplum-cc-web/./instances/demo/web"
```

becomes

```
server.document-root = "/usr/local/greenplum-cc-web-2.5.0/instances/demo/web"
```

6. Start the GPCC 2.x instance:

```
$ gpcmdr --start <instance_name>
```

At the prompt “Do you want to start the beta server?” enter `N`.

7. Change the symbolic link in your installation directory to point to your current GPCC 3.x version:

```
$ ln -sfn /usr/local/greenplum-cc-web-3.0.0 /usr/local/greenplum-cc-web
```

Managing Concurrent GPCC 2.x and 3.x Versions

The `gpcmdr` utility manages the instances for the version of GPCC in the current `gpcc_path.sh`. Assuming your .bashrc or other source file contains source `$GPPERFMONHOME/gpcc_path.sh`, to switch between versions, just modify the symbolic link `greenplum-cc-web` (in the `/usr/local` directory, by default) to point to the appropriate version.

To manage 2.x instances:

```
$ ln -sfn /usr/local/greenplum-cc-web-2.x.x /usr/local/greenplum-cc-web
```

To manage 3.x instances:

```
$ ln -sfn /usr/local/greenplum-cc-web-3.x.x /usr/local/greenplum-cc-web
```

Pivotal

Creating Greenplum Command Center Console Instances

A Command Center Console instance is a web server providing an HTML5 graphical console application to monitor system metrics and perform some administrative tasks for a single Greenplum Database cluster.

The Command Center Console runs on the gpmonws web server. The default web server port is 28080. Configuration files, log files, and runtime files for each Command Center instance are managed in a subdirectory of the `$GPPERFMON/instances` directory.

If you have multiple Greenplum Database instances, you can create separate Command Center Console instances for each of them. Each separate console instance operates on a unique port and has its own unique configuration options. A multi-cluster view may be enabled to allow you to view status for all clusters. See Enabling Multi-Cluster Support for more information.

For more information about the web server, see Web Server Administration.

The Command Center Console supports current browser versions of Chrome, Safari, Firefox, and Internet Explorer.

Before You Begin

Ensure that the following prerequisites are satisfied:

- Greenplum Command Center software is installed. See Install the Greenplum Command Center Software.
- The gpperfmon database is created and the data collection agents are running. See Creating the gpperfmon Database.
- Any certificates or Kerberos keytab files needed for encryption and user authentication are installed. See Securing a Greenplum Command Center Console Instance.
- If the `gpmon` user is to be authenticated with Kerberos, install the keytab file for the `gpmon` Kerberos principal on the Greenplum master and standby hosts and run `kinit gpmon` before you begin to create the Command Center Console instance. See Securing the gpmon Database User for more information.

The `gpcmdr --setup` command-line utility sets up the Command Center instance. The command can be run interactively, or you can create a configuration file and run the command non-interactively. If you use a configuration file, you can create multiple Command Center instances at once.

- Creating an instance interactively
- Creating an instance with a configuration file

Creating the Greenplum Command Center Instance (Interactive)

Follow the steps below to create a new Command Center Console instance. To accept the displayed default values for any parameters at configuration time, press the **ENTER** key. To monitor multiple Greenplum Database clusters, run the setup utility separately to create an instance for each cluster.

1. Log in as the Greenplum administrator (`gpadmin`) and source the `$GPPERFMON/gpcc_path.sh` file.
2. With the Greenplum Database instance running, launch the setup utility. For example:

```
$ gpcmdr --setup
```

3. Provide an instance name for the Greenplum Database instance monitored by this Console.
4. Provide a display name for the instance. This name is shown in the Console user interface. This prompt does not appear if the master host is remote.
5. Select `y` or `n` to specify if the Greenplum Database master for this instance is on a remote host. Note that Console performance is better when the Console and Greenplum Database master are on the same host. If the master host is remote, enter `y` and enter the hostname or IP address of the master at the prompt.
6. Provide the port number for the Greenplum Database master instance.
7. Provide a port number for the Command Center Console web server. The default is 28080.
8. Enter `y` to the prompt `Enable kerberos login for this instance?` to use Kerberos authentication. To use this feature, Kerberos authentication must be enabled for Greenplum Database and the Kerberos administrator must have created a keytab file for Command Center. See Enabling Kerberos Authentication with Greenplum Command Center for details. If you enter `n` you can enable Kerberos authentication later using the `gpcmdr --`

`krbenable` command.
If you choose to enable Kerberos authentication:

   a. At the prompt `Enter web server name for this instance:` enter the name of the host from the Kerberos principal. The principal name is in the format `HTTP/<host>@<realm>`. The host must be entered in the same format as the Kerberos principal and should exclude the port number.
   b. At the prompt `Enter the GPDB Kerberos service name:` enter the name of the Kerberos service principal for Greenplum Database.
   c. At the prompt `Choose Kerberos mode:` enter the number of the Kerberos mode you want to use. See Enabling Authentication With Kerberos for a description of these options.
   d. At the prompt `Enter the path to the keytab file:` enter the full path to the keytab containing the web server principal. If you are setting up this instance on the Greenplum master, the keytab file may be the same one used for Greenplum Database.

9. Enter `y` to enable SSL connections for the Command Center Console, or `n` if you do not want SSL.
   Note: Because database login information is sent over the network, it is strongly recommended to use SSL to encrypt these communications.
   You are asked to specify the location of your X509 certificate file. Enter the full path to the certificate file. The path you enter is added to the `app.conf` file. n

10. Enter `y` or `n` to specify whether you want this installation copied to a standby master. If you enter `y`, you are prompted for the standby master host name.

11. Start the Console and log in. See Connecting to the Greenplum Command Center Console.

12. You can also configure authentication so that other Greenplum users can log in to the Console, see Configuring Authentication for the Command Center Console for details.

Creating the Command Center Console Instance (Non-interactive)

It can be useful to run `gpcmdr --setup` non-interactively, taking input from a file. For example, you could install GPCC and create Command Center instances as part of a Greenplum cluster installation script. To accomplish this, create a configuration file and supply it to the `gpcmdr` utility using the `--config_file` option:

```
gpcmdr --setup --config_file file
```

The configuration file is similar to a Windows INI file, containing one or more sections beginning with a section header in square braces. Parameters in the optional `[DEFAULT]` section apply to all subsequent sections and may be overridden. Each section other than `[DEFAULT]` defines a Command Center Console instance to create.

Parameters are specified one-per-line as name-value pairs separated with equals signs (`=`) or colons (`:`). Comments begin with a number sign (`#`) or semicolon (`;`) and continue to the end of the line.

Here is an example configuration file:

```
[DEFAULT]
# defaults apply to all instances
remote_db: false
enable_copy_standby: true
standby_master_host: smdw
enable_kerberos: false
enable_ssl: true
enable_user_import_cert: true
ssl_cert_file: /etc/ssl/certs/cert.pem
enable_user_import_dhe: false
enable_reuse_dhe: true

[production]
master_hostname: mdw
instance_name: prod
display_name: Production
master_port: 5432
web_port: 28080

[development]
master_hostname: mdw
instance_name: dev
enable_copy_standby: false ; override
display_name: Development
master_port: 5532
web_port: 28090
```

See Setup Configuration File for a detailed description of the setup configuration file syntax and parameters.

Start the Command Center Console Instance

Start the Greenplum Command Center Console instance by entering:

```
gpcmdr --start
```

If you do not specify an instance name, all Command Center Console instances are started. To start a particular instance, you can specify the name of the instance. For example:

```
gpcmdr --start instance_name
```

See Administering Greenplum Command Center for a complete list of administrative commands.

Connecting to the Greenplum Command Center Console

Open the Command Center Console in a supported browser using the correct hostname and port. For example, to open a Command Center instance running on port 28080 on the local host with SSL, enter the following URL in the browser:

```
https://<master_host_name>:28080/
```

At the login prompt, enter the user name and password of a Greenplum role that has been properly configured to allow authentication to Greenplum Command Center, then click **Login**. This opens the Dashboard page of the Command Center Console, which provides a graphical system snapshot and a summary view of active queries. See the Dashboard for information about the Dashboard view.

Greenplum Command Center User Guide

The Greenplum Command Center web interface is a management tool that provides system status and query monitoring facilities for Greenplum Database administrators and users.

Command Center views allow you to instantly view the overall status of the Greenplum Database system. You can drill down to see details about hosts, database segments, queries, and CPU, memory, and disk resource utilization.

The following topics describe the information displayed on each Command Center view.

- **Dashboard**
  The Dashboard displays when you first log in to the Command Center. It shows an overview of the status of the Greenplum Database cluster the Command Center manages and provides easy access to detailed information about any aspect of system status.
- **Query Monitor**
  View current running and queued queries. Select a query to view its query text and execution plan. With proper permissions, choose queries and cancel them.
- **Host Metrics**
  View real-time statistics by server in a table format.
- **Cluster Metrics**
  View charts of current and recent statistics for all hosts (excluding master and standby).
- **History**
  View queries and historical charts of statistics for a selected time period, optionally filtered by database and user.
- **System>Segment Status**
  View a status summary for all primary and mirror segments and details for each segment.
- **System>Storage Status**
  View the current percentage disk space in use for master and segment hosts, a historical chart of segment host disk usage, and current disk usage by host.
- **Admin>Permissions**
  View permissions levels for Command Center users. Users with Admin permission can change permission levels.
- **Admin>Authentication**
  View the Greenplum Database `pg_hba.conf` host-based authentication file. Users with Admin permission can edit the file.

Dashboard

The **Dashboard** displays when you first sign in to Pivotal Greenplum Command Center. The **Dashboard** provides a quick view of the current system status, Segment Health, Queries, CPU, Memory, and Disk usage. Clicking on a panel provides more detailed information about the metric.

System Information

The following system information is displayed at the top of the page.

Uptime
The elapsed time since the Greenplum Database system was last started.

GPDB Version
The version of the Greenplum Database software the monitored cluster is running.

Connections
The number of active Greenplum Database sessions (client connections).

Last Sync
Date and time the data was last synchronized. The Command Center user interface updates views with live data every 15 seconds.

System Summary

The **Segment Health** section of the Dashboard provides a quick overview of the status of the Greenplum Database managed by this instance of the Command Center.

Clicking the **Segment Health** panel displays the Segment Status Command Center page.

Database State is the current state of the Greenplum Database system. The state can be one of the following:

- **Normal**: The database is functioning with no major errors or performance issues.
- **Segment(s) Down**: The database is in change-tracking mode or resync mode. Overall performance and system reliability is greatly reduced. See the *Pivotal Greenplum Database System Administrator Guide*for information about resolving this condition.
- **Database Unreachable**: The Greenplum Performance Monitor agent cannot connect to the database. The database is likely down. See the*Pivotal Greenplum Database System Administrator Guide* for troubleshooting information.
- **Unbalanced**: Some segments are not running in their preferred roles. That is, primaries are running as mirrors and mirrors are running as primaries, resulting in unbalanced processing.
- **Resyncing**: The database is performing a recovery or rebalance operation.

The bar graph in the **Segment Health** section shows the up or down status of all database segments in your Pivotal Greenplum Database system. A color indicator and associated number indicate the number of database segments that are currently in that particular state. Segments can have the following states:

- **Up** (Green)
- **Down** (Red)

Disk Usage Summary

This chart displays total disk usage and disk available for the Greenplum master host and segment hosts at the last synchronization. Hover over the chart to see the amount of disk used, free, and total.

Queries

This graph displays a summary view of active and queued queries for the last 60 minutes. Click on the colored dot next to the**Running** or **Queued** label to toggle the line on or off. At least one line must be visible at all times. Hover over the graph to display the number of queries for each visible line at that point in time.

CPU

This graph displays average CPU usage across the entire cluster, for the last 60 minutes. The graph displays separate lines for system processes and user processes. The user CPU usage includes the Greenplum database master, standby, and segment processes. Click on the colored dot next to the **System** or **User** label to toggle that line on or off. At least one line must be visible at all times.

Hovering the cursor over a line in the graph displays a small window with the percentage of CPU used at that point in time for the visible lines and the total if both the system and user lines are visible.

Memory

This graph displays the average percent of memory used across the entire cluster over the last 60 minutes. Hover over the line to display the percent of memory used at that point in time.

Alerts

Admin and Operator permission levels only

The **Alerts** panel displays recent messages from the Greenplum Database `pg_log` log file. The panel is updated at each synchronization. Filter the messages by severity level using the controls at the top right of the panel.

Query Monitor

The **Query Monitor** view allows you to view detailed information for active queries running on the Greenplum Database system. Users with Admin or Operator permission can see and cancel all users' queries.

Data is collected on currently running queries and the query monitor metrics are updated every 15 seconds. The time of the last update and a graphical timer showing the time remaining before the next update are displayed at the top of the page.

With the information available in this view, Greenplum Database administrators can easily:

- Understand how the system is being used — both in real-time and trending over time.
- Identify and diagnose problem queries while they are running, detect skew, find runaway queries, and so on.
- Review and balance the query load on the system by better optimizing and scheduling the query load.
- Cancel queries that disrupt system performance.

Query Metrics

The Query Monitor table displays the following columns for queries.

Query ID
: An identification string for the query. In the Console, this looks like "1295397846-56415-2". Command Center generates this ID by combining the query record's `tmid`, `ssid`, and `ccnt` fields. (See queries_* in the *Command Center Database Reference*.)

Status
: The status of the query. This can be one of the following:

  - Queued: the query has not yet started to execute
  - Running: execution has started, is not yet complete
  - Done: completed successfully
  - Cancelling: cancel request sent, cancel pending
  - Cancelled: terminated, no longer running

User
: The Greenplum Database user who submitted the query.

Database
: The name of the database that was queried.

Submit Time
: The time the query was submitted to the query planner.

Queue Time
: The amount of time the query has been (or was) in queue awaiting execution.

Run Time
: The amount of time since execution began.

CPU %
: (Active queries only.) Current CPU percent average for all processes executing this query. The percentages for all processes running on each segment are averaged, and then the average of all those values is calculated to render this metric. Current CPU percent average is always zero in historical and tail data.

CPU Skew
: The amount of CPU skew. CPU skew occurs when query executor processes for one segment use a disproportionate amount of CPU compared to processes for other segments executing the query. This value is the coefficient of variation for the CPU used by processes running this query on each segment, multiplied by 100. For example, a value of .95 is shown as 95.

Row Skew
: A measure of row skew in the system. Row skew occurs when one segment produces a disproportionate number of rows for a query. This value is the coefficient of variation for the Rows Out metric of all iterators across all segments for this query, multiplied by 100. For example, a value of .95 is shown as 95.

Queue
: The name of the resource queue for the query.

Priority
: Each query inherits the priority assigned to its resource queue. For more information about Resource Queues and Query Plans, refer to the *Greenplum Database Administrator Guide*.

Using the Query Monitor Controls

- Click a column heading to sort the rows on that column in ascending or descending order.
- Click the checkbox at the left of a row to choose a query to cancel or export. Click the checkbox in the heading row to choose all queries.
- Click **Cancel Query** to cancel selected queries.
- Click **Export** to download a comma-separated values (CSV) text file containing rows for the selected queries. When no queries are selected, all rows are exported. The default file name is `spreadsheet.csv`.
- Click any query ID to see the Query Details, including metrics, the text of the query, and the query plan.

Host Metrics

The **Host Metrics** page displays a table of the hosts in the cluster with statistics collected at the most recent quantum interval. At the top, **Last Sync** displays the time the statistics were last updated.

Click a column header to sort the table by that column. Click again to toggle between ascending and descending sort. Master and standby hosts are not included in the sort and are always displayed following the sorted list of segment hosts.

For each server, the following columns are displayed:

Hostname
: The hostname name of the server.

CPU Total/Sys/User (%)
: The total percentage of CPU in use is displayed next to a graph illustrating the CPU used for system and user processes. Hover over the table cell to show the percentages used for system and user processes and the percentage CPU idle.

Memory In Use (%)
: The percentage of host memory in use is displayed next to a graph illustrating the memory in use and available. Hover over the table cell to see memory used and available in gigabytes.

  Memory is calculated as follows:

  Total = MemTotal
  Free = MemFree + Buffers + Cached
  Used = Total - Free

Disk R (MB/s) | Skew
: Disk read rate in megabytes per second is displayed next to a graph of calculated disk read skew. Hover over the table cell to see a Low/Medium/High rating for disk skew.

Disk W (MB/s) | Skew
: Disk write rate in megabytes per second is displayed next to a graph of calculated disk write skew. Hover over the table cell to see a Low/Medium/High rating for disk write skew.

Net R (MB/s) | Skew
: Network read rate in megabytes per second is displayed next to a graph of calculated network read skew. Hover over the table cell to see a Low/Medium/High rating for network read skew.

Net W (MB/s) | Skew
: Network write rate in megabytes per second is displayed next to a graph of calculated network write skew. Hover over the table cell to see a Low/Medium/High rating for network write skew.

About Skew Calculations

Disk and Network skew ratings are calculated as each server's standard deviation from the mean calculated from all segment hosts.

Low
: Value is within 1 standard deviation from the mean. (Note: if the variance of the set is less than 3, skew is considered low regardless of deviation from mean.)

Moderate
: Value is between 1 and 2 standard deviations from the mean.

Very High
: Value is greater than 3 standard deviations from the mean.

Cluster Metrics

The **Cluster Metrics** page shows consolidated statistics for all segment hosts in the Greenplum cluster. Master and standby master hosts are excluded from the metrics.

The charts display metrics for the last time period set by the control in the top right corner of the screen.

Use the **Show/hide Charts** control to choose which metrics to display.

Hover over any of the charts to see values for the metrics at a point in time in pop-up boxes. The charts are synchronized so that hovering over any chart shows the same point in time in all charts.

The current value of a metric is shown in the upper right corner of its chart.

On charts with multiple metrics, toggle the display for a line on or off by clicking the line’s label in the legend at the top right of the chart. At least one line must be displayed. All lines are redisplayed at the next quantum interval.

The page has charts for the following metrics:

Queries
: The number of queries running and the number of queries queued to run.

CPU
: The percentage CPU used by system processes and the percentage CPU used by user processes.

Memory
: Percentage of memory in use.

  Memory is calculated as follows:

  Total = MemTotal
  Free = MemFree + Buffers + Cached
  Used = MemTotal - Free

Disk I/O
: Disk read and write rates in megabytes per second.

Network
: Network I/O read and write rates in megabytes per second. Network metrics include traffic over all NICs (network interface cards), including internal interconnect and administrative traffic.

Load
: System load average for 1-minute, 5-minute, and 15-minute periods.

Swap
: Percentage of swap space used.

Monitoring Multiple Greenplum Database Clusters

The Greenplum Command Center Multi-cluster view displays health status for multiple Greenplum Database clusters. The charts can be divided into categories.

Each Greenplum Database cluster that appears in the Multi-cluster view must have its own Command Center instance. The Multi-cluster view is hosted on a single, designated master Command Center instance. The master instance retrieves the health status data from each of the clusters' Command Center instances.

Clicking the status panel for a cluster loads that cluster's Command Center into a new browser window. If auto-login is enabled in the multi-cluster configuration file, the login screen is bypassed and the Dashboard is displayed.

Multi-cluster support is enabled by deploying a configuration file to the Command Center instance directories. See Enabling Multi-Cluster Support for instructions to set up the Multi-cluster page.

By default, any Command Center user may view multi-cluster status. Access to clusters can be restricted in the multi-cluster configuration file.

The Multi-cluster view displays the following information for each cluster.

Cluster Name
: The name for the cluster, as specified in the multi-cluster configuration file.

Database State
: The current state of the Greenplum Database cluster. The state can be one of the following:

- **Normal**: The database is functioning with no major errors or performance issues.
- **Segment(s) Down**: The database is in change-tracking mode or resync mode. Overall performance and system reliability is greatly reduced. See the *Pivotal Greenplum Database System Administrator Guide*for information about resolving this condition.
- **Database Unreachable**: The Greenplum Performance Monitor agent cannot connect to the database. The database is likely down. See the *Pivotal Greenplum Database System Administrator Guide*for troubleshooting information.
- **Unbalanced**: Some segments are not running in their preferred roles. That is, primaries are running as mirrors and mirrors are running as primaries, resulting in unbalanced processing.
- **Resyncing**: The database is performing a recoverty or rebalance operation.

Uptime
: The elapsed time since the Greenplum Database system was last started.

GPDB Version
: The version of the Greenplum Database software each monitored cluster is running.

Connections
: The number of active Greenplum Database sessions (client connections).

Active Queries
: The number queries queued or currently executing in the database.

History

The **History** page allows you to display system metrics and queries executed during a specified time period. Queries may also be filtered by database and/or user.

Set the time period to display by entering dates and times in the **From** and **To** date and time fields. You can enter dates by typing them into the date field or by choosing from the pop-up calendar. Enter 24-hour times in HH:MM format.

To restrict queries that display in the query table at the bottom of the page, enter a Greenplum database name in the **Database** field, a user name in the **User** field, or both. Filtering by database and user only affects the queries displayed in the table. The metrics displayed in charts include all activity during the selected time period.

Click **Search** to display results that match your criteria.

You can click and drag on a chart to zoom in on a time range. Click **Search** to update the query list and charts to the selected range.

Scroll charts left or right by hovering over the edge of the chart and clicking an arrow. Click ‹ or › to move in half steps. Click « or » to move in full steps.

In the query list, select or hover over a query to highlight its queued and run time in the charts.

Charts of the following metrics are available. Show or hide them at any time with the checklist at the upper right of the view.

Queries
: The number of queries running and the number of queries queued to run.

CPU
: The percentage CPU used by system processes and the percentage CPU used by user processes.

Memory
: Percentage of memory in use.

Disk I/O
: Disk read and write rates in megabytes per second.

Network
: Network I/O read and write rates in megabytes per second. Network metrics include traffic over all NICs (network interface cards), including internal interconnect and administrative traffic.

Load
: System load average for 1-minute, 5-minute, and 15-minute periods.

Swap
: Percentage of swap space used.

Query Metrics

The Query table displays queries that were active during the specified time period, including queries that started before or finished after the specified time. However, queries that are still active are not included in the table; these queries can be viewed on the Query Monitor page.

The query table has the following columns:

Query ID
: An identification string for the query. In the Console, this looks like “1295397846-56415-2”.

Status
: The final status of the query. This can be one of the following:

- Done
- Cancelled

User
: The Greenplum Database user who submitted the query.

Database
: The name of the database that was queried.

Submit Time
: The time the query was submitted to the query planner.

Queued Time
: The amount of time a query spent in the queue before it was executed.

Run Time
: The amount of time the query required to produce a result.

End Time
: The time the query completed or was cancelled.

CPU Skew
: The amount of CPU skew. CPU skew occurs when query executor processes for one segment use a disproportionate amount of CPU compared to processes for other segments executing the query. This value is the coefficient of variation for the CPU used by processes running this query on each segment, multiplied by 100. For example, a value of .95 is shown as 95.

Row Skew
: A measure of row skew in the system. Row skew occurs when one segment produces a disproportionate number of rows for a query. This value is the coefficient of variation for the Rows Out metric of all iterators across all segments for this query, multiplied by 100. For example, a value of .95 is shown as 95.

Queue
: The name of the resource queue for the query.

Priority
: Each query inherits the priority assigned to its resource queue.

For more information about Resource Queues and Query Plans, refer to the *Greenplum Database Administrator Guide*.

System

The **System** view provides system metrics for individual hosts in the Greenplum cluster and for the entire cluster. Click **Host Metrics** to display metrics for each host. Click **Cluster Metrics** to display consolidated metrics for the entire cluster.

Segment Status

Storage Status

Segment Status

The **Segment Status** page provides a health overview for the Greenplum Database segments and details for each primary and mirror segment.

Segment Summary

Greenplum Database is most efficient when all segments are operating in their preferred roles. The **Segment Summary** panel tells you the overall segment status and if any mirrors are acting as primaries.

The **Segment Summary** panel provides the following information:

Database State
: The database state can be one of the following:

- **Normal**: The database is functioning with no major errors or performance issues.
- **Segment(s) Down**: The database is in change-tracking mode or resync mode. Overall performance and system reliability is greatly reduced. See the *Pivotal Greenplum Database System Administrator Guide*for information about resolving this condition.
- **Database Unreachable**: The Greenplum Performance Monitor agent cannot connect to the database. The database is likely down. See the *Pivotal Greenplum Database System Administrator Guide*for troubleshooting information.
- **Unbalanced**: Some segments are not running in their preferred roles. That is, primaries are running as mirrors and mirrors are running as primaries, resulting in unbalanced processing.
- **Resyncing**: The database is performing a recoverty or rebalance operation.

Mirrors Acting as Primary
: The number of mirror segments acting as primary segments.

Recommended Actions
: Suggests actions to perform to restore the cluster to balance. These include:

- Recover and Rebalance
- Rebalance

These actions are executed from the command line using the `gprecoverseg` Greenplum management utility. See `gprecoverseg` in the *Pivotal Greenplum Database Utility Reference* for more information.

Total Segments
: The total number of primary and mirror segments in the Greenplum cluster.

Segment Hosts
: The total number of segment hosts in the Greenplum cluster.

Segment Health

The **Segment Health** panel contains charts for Greenplum Database segments' status, replication mode, and preferred roles.

Status
: Numbers of segments that are down and up.

Replication Mode
: A chart that shows the number of segments in each of the possible replication modes.

- Not Syncing: The primary segment and mirror segment are active and all changes to the primary segment have been copied to the mirror using a file block replication process.
- Change Tracking: If a primary segment is unable to copy changes to its mirror segment using the file replication process, it logs the unsent changes locally so they can be replicated when the mirror again becomes available. This can happen if a mirror segment goes down or if a primary segment goes down and its mirror segment automatically assumes the primary role.
- Resyncing: When a down segment is brought back up, administrators initiate a recovery process to return it to operation. The recovery process synchronizes the segment with the active primary and copies the changes missed while the segment was down.
- Synced: Once all mirrors and their primaries are synchronized, the system state becomes synchronized.

Pivotal

Preferred Roles

The red portion of the Preferred Role chart shows the numbers of segments that not operating in their preferred primary or mirror roles. If the chart is not solid green, the performance of the Greenplum cluster is not optimal.

Primary and mirror segments are distributed evenly among the segment hosts to ensure that each host performs an equivalent share of the work and primary segments and their mirror segments reside on different segment hosts. When a primary segment goes down, its mirror on another host in the cluster automatically assumes the primary role, increasing the number of primary segments running on that host. This uneven distribution of the workload will affect query performance until the down segment is restored and the segments are returned to their original, preferred, roles.

Segment Table

The table at the bottom of the **Segment Status** page contains a detailed row for every primary and mirror segment in the Greenplum Cluster. The table has the following columns for each segment:

Hostname
: The name of the segment host where the segment is running.

Address
: The network interface on the segment host for the segment.

Port
: The port number assigned to the segment.

DBID
: The unique identifier for the segment instance.

ContentID
: The content identifier for the segment, from 0 to the number of segments minus 1. A primary segment and its mirror have the same ContentID. The master and standby master, which have ContentID −1, are excluded from the table.

Status
“UP” if the segment is running, “DOWN” if the segment has failed or is unreachable.

Role
: The segment’s current role, either “primary” or “mirror”.

Preferred Role
: The segment’s intended role, either “primary” or “mirror”.

Replication Mode
: The replication status for the segment. See Segment Health for possible values.

Last Event|[Total]
: The date and time of last segment health-related activity. Click to display a list of recent events.

Storage Status

The **Storage Status** page shows current historical disk usage for Greenplum master and segment hosts.

Disk Usage Summary

You can see current disk space in use, space free, and total space in the Disk Usage Summary panel. Disk space metrics for the segment hosts (GP Segments) and the master (GP Master) are shown in separate bar charts.

The GP Segments bar chart shows combined disk space for all segments.

The GP Masters bar chart shows combined disk space for master and standby master.

Hover over either of the charts to see the space used, free, and total in gigabytes and as a percentage of the total.

GP Segments Usage History

The GP Segments Usage History panel presents a chart of percentage of disk space in use for the time period set by the control in the panel header.

Hover over the chart to see the percentage disk in use on any given point.

Storage Status Table

The Storage Status table provides current disk space usage metrics for each host and by data directory within hosts.

Admin

The **Admin** view provides the ability to manage authentication and authorization for Greenplum Command Center Console and Greenplum Database users.

- Permissions
  View the Greenplum Command Center permission level for Greenplum Database users. Administrators can change user's permission levels.
- Authentication
  View the Greenplum Database host-based authentication file (`pg_hba.conf`). Administrators can change the file.

Permission Levels for GPCC Access

The **Permissions Levels for GPCC Access** screen allows users with Operator Basic, Operator, or Admin permission to view permissions for Command Center users. Users with Admin permission can set permissions for all users.

Viewing User Permissions

Initially, all Greenplum Database login users are included in the list with their current permission levels.

- To filter by role name, enter all or part of the user's database role name in the *Role Name* field. The filter performs a simple substring search and displays users with matching role names. Click the **Role Name** label to reverse the search order.
- To filter for users with a specific permission level, choose the permission level from the **Permission Level** list.
- Role Name and Permission Level filters can be used together.
- To reset the filters, remove all text from the *Role Name* field and choose **Filter by**… from the **Permission Level** list.

Changing User Permission Levels

Users with Admin permission can change permission levels.

1. Use the **Role Name** and **Permission Level** filters to display the roles you want to change.
2. Check the box next to a role name to select the user, or check the box in the heading to select all displayed users.
3. Select the new permissions level for each user from the list in the **Permission Level** column, or select a new permission level for all selected users from the **Change Selected to**… list.

Authentication

The **System>Authentication** screen allows users with Operator Basic, Operator, and Admin permission to view the Greenplum Database host-based authentication file, `pg_hba.conf`.

Users with Admin permission can add, remove, change, and move entries in the file. The Command Center UI validates entries to ensure correct syntax. Previous versions of the file are archived so that you can restore an earlier version or audit changes.

See Authentication for an overview of user authentication options for Greenplum Database and Greenplum Command Server.

See pg_hba.conf file in the PostgreSQL documentation for a detailed description of the contents of the `pg_hba.conf` file.

Viewing the Host-Based Authentication File

Choose **Admin>Authentication** to display the content of the Greenplum Database `pg_hba.conf` file.

The `pg_hba.conf` file contains a list of entries that specify the characteristics of database connection requests and authentication methods. When Greenplum Database receives a connection request from a client, it compares the request to each entry in the `pg_hba.conf` entry in turn until a match is found. The request is authenticated using the specified authentication method and, if successful, the connection is accepted.

Editing the Host-Based Authentication File

Command Center users with the *Admin* permission can edit the `pg_hba.conf` file. Note that any changes you make are lost if you move to another screen before you save them.

- To change an existing entry, click anywhere on the entry. Edit the fields and click **Save** to save your changes, or **Cancel** to revert changes.
- To move an entry up or down in the list, click on the ▦ symbol, drag the line to the desired location, and release.
- To add a new entry to the end of the file, click **Add New Entry** at the bottom of the screen. Edit the fields and click **Save** to save your changes, or **Cancel** to abandon the new entry.
- To add a new entry after an existing entry, highlight the existing entry and click ⊕. Edit the fields and click **Save** to save your changes, or **Cancel** to abandon the new entry.
- To copy an entry, select the entry and click ⧉. A copy of the selected entry is added below the selected entry and displayed for editing. Edit the fields and click **Save** to save your changes, or **Cancel** to abandon the copy.
- To add a comment to the file, add an entry by clicking **Add New Entry** or ⊕ and then choose `#` from the `Type` list.
- To toggle an entry between active and inactive, select the line and click the **active/inactive** toggle control to the right. This action adds or removes a comment character (`#`) at the beginning of the entry.
- To remove an entry, highlight the line and click ⊖. The entry is displayed with strikethrough text. You can restore the entry by highlighting it and clicking **undelete**. The entry is permanently removed when you click **Save config and update GPDB**.
- To finish editing, click **Save config and update GPDB**. Then click **Save and Update** to save your changes or click **Cancel** to return with your edits intact.

When you select **Save and Update**, the `pg_hba.conf` file is saved and refreshed in Greenplum Database. Note that existing client connections are unaffected.

Loading a Previous Version of the Host-Based Authentication File

When you save a new version of the `pg_hba.conf` file, a copy is saved in the Greenplum Database `$MASTER_DATA_DIRECTORY/pg_hba_archive` directory as `pg_hba.conf-<timestamp>`.

To view an archived version of the `pg_hba.conf` file, click **Load versions…** and click the timestamp for the version to display.

To revert to a previous version of the file, load the previous version and then click **Save config and update GPDB**. The configuration is refreshed in Greenplum Database and saved as a new version in the archive directory.

Administering Greenplum Command Center

System administration information for the Greenplum Command Center.

- About the Command Center Installation
- Starting and Stopping Greenplum Command Center
- Administering Command Center Agents
- Administering the Command Center Database
- Administering the Web Server
- Configuring Greenplum Command Center
- Enabling Multi-Cluster Support
- Securing a Greenplum Command Center Console Instance

Starting and Stopping Greenplum Command Center

Greenplum Command Center includes the command center console and the command center agents.

Starting and Stopping Command Center Agents

Whenever the Greenplum Database server configuration parameter `gp_enable_gpperfmon` is enabled in the master `postgresql.conf` file, the Command Center agents will run and collect data. These agents are automatically stopped and started together with the Greenplum Database instance.

To disable the Command Center data collection agents, you must disable the `gp_enable_gpperfmon` parameter, and restart the Greenplum Database instance.

Starting and Stopping Command Center Console

Use the following `gpcmdr` commands to start, stop and restart Greenplum Command Center Console instances:

```
$ gpcmdr --start ["instance name"]
```

```
$ gpcmdr --stop ["instance name"]
```

```
$ gpcmdr --restart ["instance name"]
```

If you do not specify an instance name, all instances are started, stopped, or restarted at once. You can check the status of instances using:

```
$ gpcmdr --status ["instance name"]
```

Administering Command Center Agents

This topic describes basic agent administration tasks, including adding hosts and viewing agent log files.

Adding and Removing Hosts

Segment agents on new hosts are detected automatically by the master agent. Whenever `gp_enable_gpperfmon` is enabled on the master, the master monitor agent automatically detects, starts, and begins harvesting data from new segment agents.

To verify the addition of a new monitored host, you can check for the new hostname in the Greenplum Command Center Console System Metrics view. Alternately, you can query the `system_now` table for the row containing current metrics for each host. For example:

```
# SELECT * FROM system_now WHERE hostname='new_hostname';
```

Viewing and Maintaining Master Agent Log Files

Log messages for the master agent are written to the following file by default:

```
$MASTER_DATA_DIRECTORY/gpperfmon/logs/gpmmon.log
```

To change the log file location, edit the `log_location` parameter in `gpperfmon.conf`.

On the segment hosts, agent log messages are written to a `gpsmon.log` file in the segment instance's data directory. For a host with multiple segments, the agent log file is located in the data directory of the first segment, as listed in the `gp_configuration` table by dbid. If the segment agent is unable to log into this directory, it will log messages to the home directory of the user running Command Center (typically `gpadmin`).

Configuring Log File Rollover

At higher logging levels, the size of the log files may grow dramatically. To prevent the log files from growing to excessive size, you can add an optional log rollover parameter to `gpperfmon.conf`. The value of this parameter is measured in bytes. For example:

```
max_log_size = 10485760
```

With this setting, the log files will grow to 10MB before the system rolls over the log file. The timestamp is added to the log file name when it is rolled over. Administrators must periodically clean out old log files that are no longer needed.

Administering the Command Center Database

Data collected by Command Center agents is stored in a dedicated database called gpperfmon within the Greenplum Database instance. This database requires the typical database maintenance tasks, such as clean up of old historical data and periodic `ANALYZE`.

See the Command Center Database Reference section for a reference of the tables and views in the gpperfmon database.

Connecting to the Command Center Database

Database administrators can connect directly to the Command Center database (`gpperfmon`) using any Greenplum Database-compatible client program (such as `psql`). For example:

```
$ psql -d gpperfmon -h master_host -p 5432 -U gpadmin
```

Backing Up and Restoring the Command Center Database

The history tables of the Command Center database (`gpperfmon`) can be backed up and restored using the Greenplum Database parallel backup and restore utilities (`gpcrondump`, `gpdbrestore`). See the *Greenplum Database Utility Guide* for more information.

Because the Command Center database has a low number of tables, you may prefer to devise a backup plan using the table-level backup features of `gp_dump`. For example, you can create scripts to run `gp_dump` to back up the monthly partitions of the historical data tables on a monthly schedule. Alternately, you can back up your Command Center database at the database level.

Maintaining the Historical Data Tables

All of the `*_history` tables stored in the Command Center database (`gpperfmon`) are partitioned into monthly partitions. A January 2010 partition is created at installation time as a template partition (it can be deleted once some current partitions are created). The Command Center agents automatically create new partitions in two month increments as needed. Administrators must periodically drop partitions for the months that are no longer needed in order to maintain the size of the Command Center database.

See the *Greenplum Database Administrator Guide* for more information on dropping partitions of a partitioned table.

Administering the Web Server

The gpmonws web server is installed in the `www` directory of your Greenplum Command Center installation.

Configuring the Web Server

The web server configuration file is stored in `$GPPERFMONHOME/instances/instance_name/webserver/conf/app.conf`. Some of the parameters in this configuration file are set by the `gpcmdr` setup utility, including the web server port and SSL options. See the *Web Server Parameters* section of Configuration File Reference for a description of the parameters in this file.

Viewing and Maintaining Web Server Log Files

Web server access and error log messages are written to `$GPPERFMONHOME/instances/<instance_name>/webserver/logs/gpmonws.log`.

If you experience errors viewing the Greenplum Command Center Console, refer to this file for more information.

To prevent the web server log from growing to excessive size, you can set up log file rotation using `logrotate` or `cronolog`.

Configuring Greenplum Command Center

Configuration parameters for Greenplum Command Center are stored in the Agent and Console configuration files.

Agent Configuration

Changes to these files require a restart of the Greenplum Database instance (`gpstop -r`).

- `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf`
- `$MASTER_DATA_DIRECTORY/postgresql.conf`

Console Configuration

Changes to these files require a restart of Command Center Console (`gpcmdr --restart`).

- `$GPPERFMONHOME/instances/<instance_name>/conf/clusters.conf`
- `$GPPERFMONHOME/instances/<instance_name>/webserver/conf/app.conf`

See the Configuration File Reference section for a description of the configuration parameters in these files.

You should not need to manually edit any of the files. Running the Command Center setup utility will make all the necessary modifications to these configuration files.

Enabling Multi-Cluster Support

Multi-cluster functionality allows you to view the status of multiple Greenplum Database clusters at one time in the Command Center user interface. The clusters can be organized into groups.

Typically, one Greenplum cluster is designated the *master cluster*; its Command Center instance hosts the multi-cluster view for all Command Center-managed clusters.

See Monitoring Multiple Greenplum Database Clusters for information about accessing the multi-cluster view.

Setting Up Multiple Clusters

Multi-cluster support is enabled with a multi-cluster configuration file. There is a template for this configuration file in the instance directory of each Command Center instance at the following location:

```
$GPCCHOME/instances/<instance_name>/conf/clusters.conf
```

Locate the `clusters.conf` template on the Command Center instance you want to identify as the master instance. This will be the web server that hosts the multi-cluster web page.

The configuration file consists of the following values separated by colons:

```
SERVER : HOST : PORT : TABGROUP : AUTOLOGIN : SSL : ACCESS
```

For example:

```
Miracle:www.example.com:28080:Production:True:false:gpdb_role,accounting_role
Deforest:192.51.100.186:28080:Development:False:false
Grandalpha:grandalpha:32020:Development:False:false
```

All fields except the last, **ACCESS**, are required.

SERVER
: The server value is a primary key, used to uniquely identify each Greenplum Database cluster.

  The name may not contain special characters, other than the space character, underscore (`_`), or hyphen (`-`).

  Command Center throws an error if there are any entries with the same primary key.

HOST
: This is the host name or IP address for the cluster's Command Center UI.

PORT
: The Command Center port number.

TABGROUP
: This field is used to divide Greenplum Database clusters into categories; for example, **Testing**, **Production**, and **Deployment**.

AUTOLOGIN
: This field enables automatic login to the cluster from the multi-cluster view. When automatic login is enabled, clicking the cluster's health chart on the multi-cluster page loads the cluster's Command Center UI in a new browser window.

  The **AUTOLOGIN** field is a true or false value. It is not case-sensitive.

SSL
: This field indicates whether SSL is enabled for the host. It takes a true or false value and is not case-sensitive. Any other value is an error, which will be shown in the UI.

  Important: All hosts must have the same SSL configuration. SSL must be enabled or disabled for all hosts.

ACCESS
: This optional field contains a comma-separated list of database roles that can see the **Multiple GPDB Clusters** view. If this field is empty, *all* users can see the view.

Securing a Greenplum Command Center Console Instance

A Greenplum Command Center Console instance can be secured by encrypting network traffic between the web server and users' browsers, authenticating Command Center users, and managing users' permissions to access Command Center features.

SSL/TLS Encryption

Greenplum Command Center supports SSL/TLS encryption to secure connections between browsers and the Command Center web server. To enable SSL, you should have a signed certificate for the Command Center web server in place when you create the Command Center instance.

Place your certificate on the server where Command Center is installed, for example in the `/etc/ssl/certs` directory of the Greenplum master host. You import the certificate when you create a Command Center instance with the `gpcmdr --setup` command. The locations of the certificate and private key files are saved in the `$GPPERFMONHOME/instances/<instance_name>/webserver/conf/app.conf` configuration file for the command center instance. See Command Center Console Parameters for details.

You can request a certificate from your organization's internal certificate authority or a commercial certificate authority, or you can use a self-signed certificate you create yourself with a cryptography suite such as OpenSSL. If you create a self-signed certificate, note that clients will have to override a security warning when they first connect to the Command Center web server.

Authentication Options

Users logging in to Greenplum Command Center are authenticated with the Greenplum Database host-based authentication system. Users can enter credentials as a user name and password or, if Kerberos authentication is configured, by authenticating with Kerberos on their workstation before browsing to the Command Center web server.

Database users must first be added to the Greenplum Database by using commands such as `CREATE ROLE` or `CREATE USER`. The `LOGIN` privilege is required. This example creates a login user with an encrypted password:

```
CREATE ROLE cc_user WITH LOGIN ENCRYPTED PASSWORD 'changeme';
```

The `pg_hba.conf` configuration file determines how authentication will proceed. This file contains a list of entries that are compared to attributes of the user's connection request, including the type of connection, network location of the originating host, database name, and login user name. When a match is found, the authentication method specified in the entry is applied.

The `pg_hba.conf` file can be viewed by Operators and edited by Admins in the Command Center console on the Admin>Authentication page.

Users configured with local `trust` authentication are denied access to Command Center because this method is not secure. Using remote `trust` authentication is discouraged for the same reason.

The `md5` and `password` authentication methods authenticate the user name and password with the Greenplum Database `pg_roles` system table. The `md5` method requires the password to be MD5-encoded when sent over the network, so it is preferred over the `password` method, which sends the password in clear text.

The `ldap` authentication method authenticates the user name and password with an LDAP server. The LDAP server and parameters are specified in the options field of the `pg_hba.conf` entry. See the PostgreSQL LDAP authentication documentation for the format of the LDAP options.

The `gss` authentication method is used for Kerberos authentication. To use Kerberos with Command Center, Kerberos authentication must be enabled for the Greenplum Database system and the Command Center instance must also be configured. Users authenticate with the Kerberos KDC on their workstations (using `kinit`, for example) before connecting to the Command Center web server. The role name in Command Center is the user's Kerberos principal name.

For details about setting up Kerberos authentication, see Enabling Kerberos Authentication with Greenplum Command Center.

See the PostgreSQL Authentication methods documentation for additional details of the authentication options.

Authorization

Command Center manages permission levels using Greenplum Database roles and groups. The Basic, Operator Basic, and Operator permission levels correspond to the `gpcc_basic`, `gpcc_operator_basic`, and `gpcc_operator` group roles in the database. The Admin permission level is conferred to roles that have the `SUPERUSER` privilege. A user who has not been added to any of the groups and does not have `SUPERUSER` privilege has the most restrictive permission level, Self Only.

Greenplum Database superusers can manage permission levels on the Command Center Admin>Permissions page. Superusers can also directly assign users roles in the database by using the `ALTER USER`, `ALTER GROUP`, and related commands to add or remove users from groups and add or remove the `SUPERUSER` privilege. If a role is configured for more than one permission level, Command Center uses the highest permission level.

Command Center users have the following capabilities, according to their permission levels:

Self Only

Users can view metrics and view and cancel their own queries.

Any Greenplum Database user successfully authenticated through the Greenplum Database authentication system can access Greenplum Command Center with Self Only permission. Higher permission levels are required to view and cancel other's queries and to access the System and Admin Control Center screens.

Basic

Allows users to view metrics, view all queries, and cancel their own queries.

Users with Basic permission are members of the Greenplum Database `gpcc_basic` group.

Operator Basic

Allows users to view metrics, view their own and others' queries, cancel their own queries, and view the System and Admin screens.

Users with Operator Read-only permission are members of the Greenplum Database `gpcc_operator_basic` group.

Operator

Allows users to view their own and others' queries, cancel their own and other's queries, and view the System and Admin screens.

Users with Operator permission are members of the Greenplum Database `gpcc_operator` group.

Admin

Allows users access to all views and capabilities in the Command Center.

Greenplum Database users with the `SUPERUSER` privilege in Greenplum Database have Superuser permissions in Command Center.

Configuring Authentication for the Command Center Console

Greenplum Command Center users are Greenplum Database users, authenticated using the standard Greenplum Database host-based authentication system. When authentication is properly configured, a user can use the same credentials to log into a database with a database client such as `psql` and into the Command Center web interface with a browser.

To create a new Command Center user, first you have to create a Greenplum Database user, then edit the Greenplum host-based authentication configuration file (`pg_hba.conf`) to give that user access to Command Center.

Any Greenplum Database user who can authenticate via the `pg_hba.conf` file can log in to Greenplum Command Center and view or cancel their own queries and view metrics screens. A user's Command Center permission level determines if additional Command Center features are accessible. See Authorization for information about permissions.

The following are steps to create new Command Center users in an interactive `psql` session. With the exception of the `CREATE ROLE` command to create a new database user, all of these steps can be performed in the Command Center on the **Admin>Permissions** or **Admin>Authorization** screens.

See the *Greenplum Database Administrator Guide* for more detailed information about creating database users and roles.

1. Login as `gpadmin` on the master host.
2. Start `psql`:

```
$ psql
```

3. Enter the `CREATE ROLE` command to create a user:

```
# CREATE ROLE cc_user WITH LOGIN ENCRYPTED PASSWORD 'changeme';
```

To create an Admin user - a role with superuser privileges in the database and Greenplum Command Center:

```
# CREATE ROLE cc_admin WITH LOGIN ENCRYPTED PASSWORD 'changeme' SUPERUSER CREATEDB;
```

4. For users other than Admin, set the permission level by adding the user to a Command Center group role:
To create a Basic user: add the user to the `gpcc_basic` role:

```
# GRANT gpcc_basic TO cc_user;
```

To create an Operator Basic user - add the user to the `gpcc_operator_basic` role:

```
# GRANT gpcc_operator_basic TO cc_user;
```

To create an Operator user - add the user to the `gpcc_operator` role:

```
# GRANT gpcc_operator TO cc_user;
```

5. Grant permissions to a group by granting the role to the group:

```
# CREATE ROLE cc_users;
# GRANT cc_users to cc_user;
# GRANT gpcc_operator to cc_users;
```

6. Verify that roles were created successfully using the following command:

```
# \du
```

The new users you created are returned along with the attributes you specified.

7. Edit the `pg_hba.conf` file to give new users access to databases and the Command Center. Open the file in an editor:

```
$ vi $MASTER_DATA_DIRECTORY/pg_hba.conf
```

8. Scroll to the bottom of the file and insert the following lines to give the new users access from any IP address using password authentication:

```
host     gpperfmon    cc_user    127.0.0.1/28     md5
host     gpperfmon    cc_admin   127.0.0.1/28     md5
```

List additional databases the users can access after `gpperfmon`, or replace `gpperfmon` with `all` to allow the users to access any database.
Note: If you subsequently have issues logging in to Command Center it may be due to your specific environment; check the `$GPPERFMON/instances/instance_name/logs/gpmonws.log` log file for authentication errors. Edit the `pg_hba.conf` file based on the error message and your specific environment.

9. Save the file and exit the editor.

10. Enter the following command to reload Greenplum Database processes.

```
# gpstop -u
```

Enabling Authentication with Kerberos

If you have enabled Kerberos authentication for Greenplum Database, you can set up Greenplum Command Center to accept connections from Kerberos-authenticated users.

Greenplum Database and Command Center include support for the Generic Security Service Applications Program Interface (GSS-API) standard. A related standard, Simple and Protected GSS-API Negotiation Mechanism (SPNEGO), describes the protocol GSS-API clients and servers use to agree on the method of authentication.

With a SPNEGO-compliant web application such as Command Center, the client and server agree on the authentication method on the client's initial HTTP request. If Kerberos authentication is not supported on both ends of the connection the server falls back to basic authentication, and displays a login form requesting a user name and password. If a user has authenticated on the workstation with Kerberos and has a valid ticket granting ticket, the web browser offers the user's credential to the Command Center web server. A Kerberos-enabled Command Center web server is configured to handle the authenticated user's connection request in one of three modes, called strict, normal, or gpmon-only.

Strict
: Command Center has a Kerberos keytab file containing the Command Center service principal and a principal for every Command Center user. If the principal in the client's connection request is in the keytab file, the web server grants the client access and the web server connects to Greenplum Database using the client's principal name. If the client's principal is not in the keytab file, the server falls back to basic authentication.

Normal
: The Command Center Kerberos keytab file contains the Command Center principal and may contain principals for Command Center users. If the principal in the client's connection request is in Command Center's keytab file, it uses the client's principal for database connections. Otherwise, Command Center uses the `gpmon` user for database connections.

gpmon-only
: The Command Center uses the `gpmon` database role for all Greenplum Database connections. No client principals are needed in the Command Center's keytab file.

If you have set up Kerberos authentication for Greenplum Database, most of the configuration required to enable Command Center Kerberos authentication has been done. The Command Center Kerberos configuration builds upon the Greenplum Database Kerberos setup.

Kerberos authentication can be enabled by responding to prompts when you set up a new Command Center instance with the `gpcmdr --setup` command, or you can use the `gpcmdr --krbenable <instance-name>` command to enable Kerberos for an existing Command Center instance.

Before You Begin

Kerberos authentication must be enabled for Greenplum Database. See Using Kerberos Authentication for instructions. Make sure the following prerequisites are met before you continue:

- The `krb5-workstation` package and associated libraries (`libkrb5*`) must be installed on the Greenplum master host and each client workstation.
- The date and time on the Greenplum master host and all client workstations must be synchronized with the KDC.
- The `krb5.conf` configuration file must be the same on the KDC host, the Greenplum master host, and client workstations.
- The KDC database must have a service principal for Greenplum Database. The default service name for Greenplum Database is `postgres/<master-host>@<realm>` . You can choose a service name other than `postgres` , but it must match the value of the `krb_srvname` parameter in the `$MASTER_DATA_DIRECTORY/postgresql.conf` file.
- A keyfile file with the Greenplum Database principal must be installed on the Greenplum master host and identified by the `krb_server_keyfile` parameter in the `$MASTER_DATA_DIRECTORY/postgresql.conf` file.
- Each client workstation must have a keytab file containing their Kerberos principal, `<username>@<realm>` .

Add Command Center Principals to the KDC Database

Before you configure a Command Center instance for Kerberos authentication, you must create the required Kerberos principals. All of the principals used with Command Center are created in the Greenplum Database Kerberos realm. Command Center users use the same Kerberos principal to log in to Command Center and Greenplum Database.

Command Center Service Principal

A service principal is needed for the Command Center web server. This principal has the format `HTTP/<host>@<realm>`. For example, if users access Command Center at the URL `http://mdw.example.com:28080`, the `<host>` part of the service key is `mdw.example.com` and the `<realm>` part is the Greenplum Database Kerberos realm, for example `GPDB-KRB.EXAMPLE.COM`.

Note that Kerberos authentication only works if Command Center users enter the host in the same format specified in the Kerberos service principal. If the principal specifies the FQDN, for example, using the host's IP address in the browser URL will not work; the web server will fall back to basic authentication, presenting a login screen.

Greenplum Database gpmon User

Command Center uses the `gpmon` Greenplum role to access the `gpperfmon` database, which contains data presented in the Command Center UI.

You can choose to authenticate the `gpmon` user with Kerberos or with basic authentication. To use Kerberos, you must create a principal for the `gpmon` user.

If you choose to use basic authentication you do not need a Kerberos principal for the `gpmon` user. The `gpmon` user will authenticate with Greenplum Database using the password saved in the `~gpadmin/.pgpass` file on the host running the Command Center instance. See Changing the gpmon Password for instructions to manage the `gpmon` password.

Command Center Users

Add Kerberos principals for any Command Center users who do not already have principals in the KDC for Greenplum Database.

Adding Kerberos Principals

To add the required principals, perform the following steps as root on the KDC server.

1. Start `kadmin.local`.

   ```
   kadmin.local
   ```

2. Add a principal for the Command Center web service. Be sure to specify the `<gpcc-host>` in the same format that users should enter the host in their browsers.

   ```
   kadmin.local: addprinc HTTP/<gpcc-host>@<realm>
   ```

3. If you want the `gpmon` database user to use Kerberos authentication, add a `gpmon` principal.

   ```
   kadmin.local: addprinc gpmon@<realm>
   ```

4. Add principals for any new Command Center users.

   ```
   kadmin.local: addprinc cc_user1@<realm>
   ```

   Repeat for each new Command Center user.

5. Enter `quit` to exit `kadmin.local`.

Set Up Keytab Files

After you have created all of the Kerberos principals needed, you create and distribute keytab files. Keytab files contain Kerberos principals and encrypted keys based on the principals' Kerberos passwords. Keytab files are needed for Greenplum Database, the Command Center instance, and each Command Center and Database user.

The Command Center instance is usually installed on the Greenplum master and, when this is true, a single keyfile file can be shared by Greenplum Database and the Command Center instance. Running Command Center on the Greenplum master is recommended, since it confines authentication with the database to a single host.

If you install the Command Center instance on a host other than the Greenplum master, you will need to create a separate keyfile file.

You must also create a keyfile file for each Greenplum Database or Command Center user containing just the user's principal. This keyfile file is installed on the user's workstation to enable the user to authenticate to Kerberos.

Command Center Instance on the Greenplum Master Host

If the Greenplum Command Center web server is running on the Greenplum Database master host, Command Center can share the Greenplum Database keyfile file. You need to create a keyfile file that contains the following principals:

- Service key for the `postgres` process on the Greenplum Database master host, for example `postgres/mdw.example.com@GPDB.EXAMPLE.COM` .
- Service key created for Command Center in the previous section, for example `HTTP/mdw.example.com@GPDB.EXAMPLE.COM.`
- A principal for every Kerberos-authenticated Greenplum Database or Command Center user.

All service keys and principals should be in the Greenplum Database realm.

To create a keytab file for Greenplum Database and Command Center, perform perform the following steps as root on the KDC server.

1. Start `kadmin.local`.

   ```
   kadmin.local
   ```

2. Create a keytab file and add the Greeplum Database service key, the command center service key, and all database and Command Center users.

   ```
   kadmin.local: ktadd -k gpdb-kerberos.keytab postgres/mdw.example.com@GPDB.EXAMPLE.COM HTTP/mdw.example.com@GPDB.EXAMPLE.COM
   ```

   You can enter one or more principals with each `ktadd` command. You can specify a wildcard using the `-glob` option. For example this command adds all principals in the `GPDB.EXAMPLE.COM` realm, including service principals and admin users.

   ```
    kadmin.local: ktadd -k gpdb-kerberos.keytab -glob *@GPDB.EXAMPLE.COM
   ```

3. Enter `quit` to exit `kadmin.local`.

4. Copy the keyfile you created to the Greenplum Database master host, replacing the old keytab file. The location of the file is given by the `krb_server_keyfile` parameter in the `$MASTER_DATA_FILE/postgresql.conf` file. Set the permissions on the file so that it can be read only by the `gpadmin` user.

5. Update any entries required for new Greenplum Database principals in the `pg_hba.conf` file and `pg_ident.conf` files. See Update the Greenplum Database pg_hba.conf File for details.

Command Center Instance on a Separate Host

If the Command Center web server is on a different host than the Greenplum Database master, you need separate keytab files for Greenplum Database and Command Center. The keytab file for Greenplum Database may not require any updates, but you will need to create a keytab file for Command Center.

- The Greenplum Database keytab file must contain the Greenplum Database service key and all principals for users with database access.
- The Command Center keytab file contains the Command Center service key and principals for users that have Command Center access. Users with Command Center access must also have Greenplum Database access, so user principals in the Command Center keytab file must also be in the Greenplum Database keytab file.

Update the Greenplum Database keyfile if you created new database roles and principals for Command Center. For example, if you want to use Kerberos authentication for the `gpmon` user, you must create a principal and add it to both the Greenplum Database and Command Center keytab files.

To create the keytab file for Command Center, perform the following steps as root on the KDC host.

1. Start `kadmin.local`.

   ```
   kadmin.local
   ```

2. Create a keytab file and add the Command Center service key.

   ```
   kadmin.local: ktadd -k gpcc-kerberos.keytab HTTP/mdw.example.com@GPDB.EXAMPLE.COM
   ```

3. If you want to authenticate the `gpmon` user with Kerberos, add the `gpmon` principal.

   ```
   kadmin.local: ktadd -k gpcc-kerberos.keytab gpmon@GPDB.EXAMPLE.COM
   ```

4. Add principals for all Command Center users:

   ```
   kadmin.local: ktadd -k gpcc-kerberos.keytab cc_user1@GPDB.EXAMPLE.COM cc_user2@GPDB.EXAMPLE.COM
   ```

   You can enter one or more principals with each `ktadd` command.

5. Enter `quit` to exit `kadmin.local`.

6. Copy the keyfile you created to the the host running Command Center, for example:

   ```
   $ scp gpcc-kerberos.keytab gpadmin@<host-name>:/home/gpadmin
   ```

7. Update any entries required for new principals in the `pg_hba.conf` file and `pg_ident.conf` files on the Greenplum master. See Update the Greenplum Database pg_hba.conf File.

Update the Greenplum Database pg_hba.conf File

The Greenplum Database `$MASTER_DATA_DIRECTORY/pg_hba.conf` configuration file determines which authentication methods to use to allow database access.

If you created new Command Center users, you may need to add an entry to allow access via Command Center. The entry for an individual user has this format:

```
host database <user-name> <gpcc CIDR> gss [options]
```

Authentication for the `gpmon` user needs to be set up in the `pg_hba.conf` file in one of the following ways.

Basic authentication

> The `/home/gpadmin/.gpass` file contains the password for `gpmon` to use. See Changing the gpmon Password for details. An entry in the `pg_hba.conf` file specifies the md5 authentication method for `gpmon`:
>
> ```
> local all gpmon md5
> ```

Trust authentication

> On the Greenplum Database host only, the `gpmon` user can access databases without authentication:
>
> ```
> local all gpmon trust
> ```
>
> The `/home/gpadmin/.pgpass` file is not needed.

Kerberos authentication

> A Kerberos principal has been created for the `gpmon` user and added to the Greenplum Database and Command Center keytab files.
>
> ```
> host all gpmon <gpcc CIDR>] gss [options]
> ```
>
> Remove any existing reject rules for `gpmon`:
>
> ```
> host all gpmon <auth-method> reject
> ```

See Using Kerberos Authentication for more information about the `pg_hba.conf` file.

Enable Kerberos for the Command Center Instance

Set up the Command Center instance to use the Command Center keytab file you created.

If you are creating a *new* Command Center instance with the `gpcmdr --setup` command, answer `Y` to the prompt `Enable kerberos login for this intance?`, and enter the Command Center host name and path to the keytab file at the prompts. See Create the Greenplum Command Center Instance for complete instructions.

If you are adding Kerberos authentication to an existing Command Center instance, use the `gpcmdr --krbenable <instance-name>` command. For example, if your Command Center instance is named `my-gpcc` enter this command:

```
$ gpcmdr --krbenable my-gpcc
```

Enter the Command Center host name and path to the keytab file at the prompts. See the gpcmdr Reference for more information.

Authenticating With Kerberos on the Client Workstation

To use Kerberos Command Center authentication, the user must have authenticated with Kerberos using the `kinit` command-line tool.

The user then accesses the Command Center web server with a URL containing the host name in the format specified in the Command Center service principal and the port number, for example `http://gpcc.example.com:28080`.

The web browser must be configured to use the SPNEGO protocol so that it offers the user's Kerberos principal to the web browser. The method for configuring web browsers varies with different browsers and operating systems. Search online to find instructions to set up your browser and OS.

Securing the gpmon Database User

The Greenplum Database `gpmon` user is a superuser role used to manage the `gpperfmon` database. The `gpperfmon_install` utility, which must be run once before you can create a Command Center Console instance, creates the `gpmon` role.

Greenplum Database uses the `gpmon` role to update the `gpperfmon` database with data collected by agents running on the segment hosts. The Command Center web server uses the `gpmon` role to connect to the `gpperfmon` database as well as databases monitored by the Command Center instance.

When `gpperfmon_install` creates the `gpmon` role, it prompts for a password, which it then adds to the `.pgpass` file in the `gpadmin` user's home directory. The entry in the `.pgpass` file is similar to the following:

```
*:5432:gpperfmon:gpmon:changeme
```

See The Password File in the PostgreSQL documentation for details about the `.pgpass` file.

In the `$MASTER_DATA_DIRECTORY/pg_hba.conf` authentication file, `gpperfmon_install` creates two entries:

```
local   gpperfmon       gpmon       md5
host    all       gpmon       127.0.0.1/28    md5
```

If you authenticate users with Kerberos, you can also set up Kerberos authentication for the `gpmon` role on the Greenplum master and standby hosts. Kerberos authentication is supported with TCP connections only; `local` entries use Linux sockets and authenticate with the `.pgpass` file password, even if you have enabled Kerberos for `host` entries.

Changing the gpmon Password

To change the `gpmon` password, follow these steps:

1. Log in to Greenplum Database as a superuser and change the `gpmon` password with the `ALTER ROLE` command:

   ```
   # ALTER ROLE gpmon WITH ENCRYPTED PASSWORD 'new_password';
   ```

2. Update the password in the `.pgpass` file in the `gpadmin` home directory (`~/.pgpass`). Replace the existing password in the line or lines for `gpmon` with the new password.

   ```
   *:5432:gpperfmon:gpmon:new_password
   ```

3. Ensure that the `.pgpass` file is owned by `gpadmin` and RW-accessible by `gpadmin` only.

   ```
   $ chown gpadmin:gpadmin ~/.pgpass
   $ chmod 600 ~/.pgpass
   ```

4. Restart Greenplum Command Center with the `gpcmdr` utility.

   ```
   $ gpcmdr --restart
   ```

Authenticating gpmon with Kerberos

If you authenticate Greenplum Database and Command Center users with Kerberos, you can also authenticate the `gpmon` user with Kerberos.

1. On the KDC, create a keytab file containing the Kerberos principal for the `gpmon` user, just as you would for any Kerberos-authenticated client. Install the file on the Greenplum master and standby hosts.

2. Update the entries for `gpmon` in the `$MASTER_DATA_DIRECTORY/pg_hba.conf` file to use the `gss` authentication method.

```
host all gpmon 0.0.0.0/0 gss include_realm=0 krb_realm=GPDB.EXAMPLE.COM
```

Note that `local` entries in `pg_hba.conf` cannot be authenticated with Kerberos. If there is a `local` entry for the `gpmon` user, it will use the `.pgpass` file to authenticate with the database. See The pg_hba.conf file in the PostgreSQL documentation for complete `pg_hba.conf` file documentation.

1. Log in to the master host as `gpadmin` and authenticate the `gpmon` user.

```
$ kinit gpmon
```

1. Create the Kerberos-enabled Command Center Console instance. See Creating Greenplum Command Center Console Instances for steps to create an instance.

Utility Reference

Reference information for the two Greenplum Command Center utility programs: the `gpperfmon_install` utility that enables the data collection agents and the `gpcmdr` utility that sets up and manages the web application.

- gpperfmon_install
- gpcmdr

gpperfmon_install

Installs the Command Center database (gpperfmon) and optionally enables the data collection agents.

```
gpperfmon_install
  [--enable --password gpmon_password --port gpdb_port]
  [--pgpass path_to_file]
  [–verbose]

gpperfmon_install --help | -h | -?
```

–enable	In addition to creating the `gpperfmon` database, performs the additional steps required to enable the Command Center data collection agents. When `--enable` is specified the utility will also create and configure the gpmon superuser account and set the Command Center server configuration parameters in the `postgresql.conf` files.
–password *gpmon_password*	Required if `--enable` is specified. Sets the password of the gpmon superuser
–port *gpdb_port*	Required if `--enable` is specified. Specifies the connection port of the Greenplum Database master.
–pgpass *path_to_file*	Optional if `--enable` is specified. If the password file is not in the default location of `~/.pgpass`, specifies the location of the password file.
–verbose	Sets the logging level to verbose.
–help | -h | -?	Displays the online help.

Description

The `gpperfmon_install` utility automates the steps to enable the Command Center data collection agents. You must be the Greenplum system user (`gpadmin`) to run this utility. If using the `--enable` option, the Greenplum Database instance must be restarted after the utility completes.

When run without any options, the utility will just create the Command Center database (gpperfmon). When run with the –enable option, the utility will also run the following additional tasks necessary to enable the Command Center data collection agents:

1. Creates the `gpmon` superuser role in Greenplum Database. The Command Center data collection agents require this role to connect to the database and write their data. The `gpmon` superuser role uses MD5-encrypted password authentication by default. Use the –password option to set the `gpmon` superuser's password. Use the `--port` option to supply the port of the Greenplum Database master instance.

2. Updates the `$MASTER_DATA_DIRECTORY/pg_hba.conf` file. The utility adds the following lines to the host-based authentication file (`pg_hba.conf`). This allows the `gpmon` user to locally connect to any database using MD5-encrypted password authentication:

   ```
   local    gpperfmon   gpmon                  md5
   host     all         gpmon    127.0.0.1/28  md5
   ```

3. Updates the password file (`.pgpass`). In order to allow the data collection agents to connect as the `gpmon` role without a password prompt, you must have a password file that has an entry for the `gpmon` user. The utility adds the following entry to your password file (if the file does not exist, the utility creates it):

   ```
   *:5432:gpperfmon:gpmon:gpmon_password
   ```

   If your password file is not located in the default location (`~/.pgpass`), use the `--pgpass` option to specify the file location.

4. Sets the server configuration parameters for Command Center. The following parameters must be enabled in order for the data collection agents to begin collecting data. The utility will set the following parameters in the `postgresql.conf` configuration files:
   - `gp_enable_gpperfmon=on` (in all `postgresql.conf` files)
   - `gpperfmon_port=8888` (in all `postgresql.conf` files)
   - `gp_external_enable_exec=on` (in the master `postgresql.conf` file)

Examples

Create the Command Center database (gpperfmon) only:

```
$ su - gpadmin
$ gpperfmon_install
```

Create the Command Center database (gpperfmon), create the gpmon superuser, and enable the Command Center agents:

```
$ su - gpadmin
$ gpperfmon_install --enable --password changeme --port 5432
$ gpstop -r
```

gpcmdr

Configures and manages instances of the Command Center Console.

```
gpcmdr [--ssh_full_path <path>]
        --setup [[<section_header>] --config_file <path>]
      | --start [<instance_name>]
      | --stop [<instance_name>]
      | --restart [<instance_name>]
      | --migrate [<instance_name>]
      | --remove [<instance_name>]
      | --status [<instance_name>]
```

Type	Description
`--setup`	Configures console components on the installation host. With this option, `gpcmdr` prompts for values to configure the components and writes the values to `app.conf`. For more information on these configuration parameters, see Configuration File Reference.
`--config_file`	Sets the path to a configuration file to use to set up new Command Center instances. This option must be used with the `--setup` option. See Setup Configuration File for information about the format and content of this configuration file. If *section_header* is supplied, `gpcmdr` only sets up the instance defined in the named section in the configuration file. Otherwise, `gpcmdr` sets up all instances in the configuration file.
`--start`	Starts the specified instance (or all instances by default) and its associated web service.
`--stop`	Stops the specified instance (or all instances by default) and its associated web service.
`--migrate`	Copies Command Center instances (or all instances by default) from a previous installation.
`--remove`	Removes the specified instance and its associated database schema.
`--restart`	Restarts the specified instance (or all instances by default) and its associated web service.
`--status`	Displays the status, either `Running` or `Stopped`, of the web service.
`--version`	Displays the version of the `gpcmdr` utility.
`--ssh_full_path`	Sets the full path to the ssh command. Use this to override the ssh command found on the path.

Description

The `gpcmdr` utility sets up and configures Command Center Console instances, starts and stops instances, and provides status information.

You can set up a new Command Center Console instance interactively or, by providing a configuration file, non-interactively.

For actions `--start`, `--stop`, `--restart`, `--migrate`, and `--status` you can specify a console instance name. If you do not specify a name, the action applies to all existing console instances.

On the `--start` option, `gpcmdr` creates the `gpcc_basic`, `gpcc_operator_basic`, and `gpcc_operator` database roles if they do not already exist.

The `--migrate` option prompts you to enter the path to the Command Center installation with instances you want to migrate. The utility checks whether the instance to copy already exists in the new location before copying. If the instance exists in the new location, a prompt asks whether you want to overwrite the instance.

Examples

Interactively create a new Command Center Console instance:

```
$ gpcmdr --setup
```

Set up the Command Center Console instance defined in the `[development]` section of a configuration file:

```
$ gpmcdr --setup development gpccinstances.cfg
```

Check the status of all Command Center Console instances:

```
$ gpcmdr --status
```

Pivotal

Configuration File Reference

References for Greenplum Command Center configuration files.

Configuration parameters for Greenplum Command Center are stored in the following files:

`$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf`
: Stores configuration parameters for the Greenplum Command Center agents.

`$GPPERFMONHOME/instances/<instance_name>/webserver/conf/app.conf`
: Stores configuration parameters for the Command Center web application and web server.

`$MASTER_DATA_DIRECTORY/postgresql.conf`
: Stores configuration parameters to enable the Greenplum Command Center feature for Greenplum Database server.

`$GPPERFMONHOME/bin/ssh-wrapper`
: Greenplum Command Center normally finds the `ssh` command on the path. If your environment has an incompatible implementation of this command on the path, you can provide the absolute path to your version in the `ssh-wrapper` script, located at `$GPPERFMONHOME/bin/ssh-wrapper`.

  For example:

```
ssh="/home/me/bin/myssh"
```

Any system user with write permissions to these directories can edit these configuration files.

Command Center Agent Parameters

The `$MASTER_DATA_DIRECTORY/gpperfmon/conf/gpperfmon.conf` file stores configuration parameters for the Command Center agents. For configuration changes to these options to take effect, you must save `gpperfmon.conf` and then restart Greenplum Database server (`gpstop -r`).

To enable the Command Center agents within Greenplum Database server, you must also set the Greenplum Database Server Configuration Parameters, see Command Center Database Reference for details.

log_location
: Specifies a directory location for Command Center log files. Default is `$MASTER_DATA_DIRECTORY/gpperfmon/logs`.

min_query_time
: Specifies the minimum query run time in seconds for statistics collection. Command Center logs all queries that run longer than this value in the queries_history table. For queries with shorter run times, no historical data is collected. Defaults to 20 seconds.

  If you know that you want to collect data for all queries, you can set this parameter to a low value. Setting the minimum query run time to zero, however, collects data even for the numerous queries run by Command Center itself, creating a large amount of data that may not be useful.

min_detailed_query_time
: Specifies the minimum iterator run time in seconds for statistics collection. Command Center logs all iterators that run longer than this value in the iterators_history table. For iterators with shorter run times, no data is collected. Minimum value is 10 seconds.

  This parameter's value must always be equal to, or greater than, the value of `min_query_time`. Setting `min_detailed_query_time` higher than `min_query_time` allows you to log detailed query plan iterator data only for especially complex, long-running queries, while still logging basic query data for shorter queries.

  Given the complexity and size of iterator data, you may want to adjust this parameter according to the size of data collected. If the `iterators_*` tables are growing to excessive size without providing useful information, you can raise the value of this parameter to log iterator detail for fewer queries.

max_log_size
: This parameter is not included in gpperfmon.conf, but it may be added to this file for use with Greenplum Command Center.

  To prevent the log files from growing to excessive size, you can add the `max_log_size` parameter to `gpperfmon.conf`. The value of this parameter is measured in bytes. For example:

```
max_log_size = 10485760
```

  With this setting, the log files will grow to 10MB before the system rolls over to a new log file.

partition_age
: The number of months that Greenplum Command Center statistics data will be retained. The default it is 0, which means we won't drop any data.

quantum
: Specifies the time in seconds between updates from Command Center agents on all segments. Valid values are 10, 15, 20, 30, and 60. Defaults to 15 seconds.

  If you prefer a less granular view of performance, or want to collect and analyze minimal amounts of data for system metrics, choose a higher quantum. To collect data more frequently, choose a lower value.

ignore_qexec_packet
: When set to true, Greenplum Command Center agents do not collect performance data in the `gpperfmon` database `queries_*` tables: `rows_out`, `cpu_elapsed`, `cpu_currpct`, `skew_cpu`, and `skew_rows`. The default setting, true, reduces the amount of memory consumed by the gpmmon process. Set this parameter to false if you require this additional performance data.

smdw_aliases
: This parameter allows you to specify additional host names for the standby master. For example, if the standby master has two NICs, you can enter:

```
smdw_aliases= smdw-1,smdw-2
```

  This optional fault tolerance parameter is useful if the Greenplum Command Center loses connectivity with the standby master. Instead of continuously retrying to connect to host smdw, it will try to connect to the NIC-based aliases of `smdw-1` and/or `smdw-2`. This ensures that the Command Center Console can continuously poll and monitor the standby master.

Command Center Console Parameters

Each instance of the Command Center Console has a configuration file located at `$GPPERFMONHOME/instances/<instance_name>/webserver/conf/app.conf` .

After editing this file, reload the configuration by restarting the Command Center Console instance (`gpperfmon --restart <instance_name>`).

`appname = gpmonws`
: The web server binary file. Do not change.

`listentcp4 = [true | false]`
: When `true` , the address type is tcp4. The default is `true` .

`runmode = [prod | dev | test]`
: The application mode, which can be `dev` , `prod` or `test` . The default is `dev` . In `dev` mode Command Center shows user friendly error pages. User friendly error pages are not rendered in `prod` mode.

`session = [true | false]`
: Use sessions to manage user experience. The default is `true` . Sessions are stored in memory.

`enablexsrf = [true | false]`
: Enable CSRF protection.

`xsrfexpire = <seconds>`
: CSRF expire time. The default is `43200` seconds.

`xsrfkey = <token_string>`
: The CSRF token.

`rendertype = json`
: The render type of web server. Do not change.

`printallsqls = [true | false]`
: Print all backend gpperfmon SQL to the web server console. The default is `false` .

`sessionname = webserver_gpperfmon_instance_<instance_name>`
: The name of the session in Greenplum Database. Do not change.

`display_name = <display_name>`
: The display name for console.

`master_host = <hostname>`
: The Greenplum Database host name. The default is `localhost` .

`master_port = <port>`
: The Greenplum Database master port. The default is `5432` .

`HTTPSCertFile = </path/to/cert.pem>`
: The full path to the server's SSL certificate, if SSL is enabled.

`HTTPSKeyFile = </path/to/cert.pem>`
: The server's private key file if SSL is enabled.

`EnableHTTPS = [true | false]`
: Enable listening on the secure SSL port. The default is `true` .

`EnableHTTP = [true | false]`
: Enable listening on the HTTP port. Default is `false` .

`httpsport = [port]`
: The web server port. The default is 28080.

Setup Configuration File

A setup configuration file contains properties used to define one or more Greenplum Command Center instances when `gpcmdr` is run with the `--config_file` option.

The configuration file uses the Python configuration file format, similar to the Microsoft INI file format. The file contains sections, introduced by a `[section]` header, and followed by `name: value` or `name=value` entries, one per line. Comments begin with a `#` or `;` character and continue through the end of the line. A `[DEFAULT]` section sets default values for parameters that may be overriden in other sections.

See Setting Up Command Center Instances with a Configuration File for more information.

Parameters

`remote_db`
: `True` if the instance is to run on a different host. Default: `False`.

`master_host`
: The name of the host where the Greenplum Command Center Console is to be set up, if `remote_db` is `True`.

`instance_name`
: The name of the instance to set up. This will become the name of a subdirectory in the `instances` directory where the instances configuration and log files are stored. Instance names may contain letters, digits, and underscores and are not case sensitive.

`display_name`
: The name to display for the instance in the Command Center user interface. Display names may contain letters, digits, and underscores and *are* case sensitive. `instance_name` is used for `display_name` if this parameter is not provided.

`master_port`
: The Greenplum Database master port. Default: `5432`.

`webserver_port`
: The listen port for the Command Center go web server. This port must be different for each Command Center Console instance on the host. Default: `28080`.

`enable_ssl`
: `True` if client connections to the Command Center web server should be secured with SSL. Default: `False`.

`enable_user_cert`
: `True` if the server certificate is supplied by the user. If `False` (default) and `enable_ssl` is `True`, `gpcmdr` generates a certificate during setup. Data for the certificate's CN is entered interactively during setup. Default: `False`.

`ssl_cert_file`
: If `enable_user_cert` is `True`, set this parameter to the full path to a valid certificate in PEM file format.

`enable_kerberos`
: Set to `True` to enable Kerberos authentication.

`webserver_url`
: The web server hostname, from the Kerberos HTTP service principal.

`keytab`
: Path to the keytab file containing Kerberos principals for the Command Center web server and users.

`enable_copy_standby`
: Set to `True` to have `gpcmdr` install the instance configuration on the Greenplum standby master host.

`standby_master_host`
: The name of the Greenplum standby master host. Required when `enable_copy_standby` is `True`.

Examples

This example configuration sets up two Command Center instances, `prod` and `dev`. Parameters in the `[DEFAULT]` section apply to all instances and may be overridden by parameters in the `[production]` and `[development]` sections.

```
[DEFAULT]
remote_db=false
master_port=5432
#You need to set 'enable_user_cert' to true to import your own pem file
enable_user_cert=true
ssl_cert_file=/tmp/cert.pem

[development]
instance_name=development
enable_copy_standby=false
webserver_port=28080
enable_ssl=false

[production]
instance_name=production
display_name=OurProduction
remote_db=true
master_host=mdw
master_port=15432
webserver_port=28081
enable_ssl=true
enable_user_cert=true
enable_copy_standby=true
standby_master_host=smdw
```

This example configuration has only one instance, so the `[DEFAULT]` section and section headers are not necessary.

```
instance_name=single
display_name=SINGLE
remote_db=true
master_host=10.152.10.149
master_port=5432
webserver_port=28082
enable_ssl=true
enable_user_cert=true
ssl_cert_file=/tmp/cert.pem
enable_copy_standby = true
standby_master_host=192.0.2.156
```

Greenplum Database Server Configuration Parameters

The following parameters must be uncommented and set in the server configuration file (`postgresql.conf`) in order to enable the Command Center data collection agents:

- `gp_enable_gpperfmon` and `gpperfmon_port` must be set in both the master and segment `postgresql.conf` files.
- `gp_enable_gpperfmon` and `gp_enable_gpperfmon` only need to be set in the master `postgresql.conf` file.

After changing these settings, the Greenplum Database instance must be restarted for the changes to take effect.

gp_enable_gpperfmon
: Turns on the Command Center data collection agent for a segment. Must be set in all `postgresql.conf` files (master and all segments).

gpperfmon_port
: The default port for the Command Center agents is 8888, but you can set this parameter to a different port if required (master and all segments).

gp_gpperfmon_send_interval
: Sets the frequency in seconds that the Greenplum Database server processes send query execution updates to the Command Center agent processes.

gp_external_enable_exec
: This parameter is enabled by default and must remain enabled. It allows the use of external tables that execute OS commands or scripts on the segment hosts. The Command Center agents use this type of external tables to collect current system metrics from the segments.

gpperfmon_log_alert_level
: Controls which message levels are written to the gpperfmon log. Each level includes all the levels that follow it. The later the level, the fewer messages are sent to the log. The default value is warning.

Pivotal

Command Center Database Reference

References for the Greenplum Command Center `gpperfmon` database tables.

The Command Center database consists of three sets of tables; `now` tables store data on current system metrics such as active queries, `history` tables store data on historical metrics, and `tail` tables are for data in transition. `Tail` tables are for internal use only and should not be queried by users. The `now` and `tail` data are stored as text files on the master host file system, and accessed by the Command Center database via external tables. The `history` tables are regular database tables stored within the Command Center (`gpperfmon`) database.

The database consists of three sets of tables:

- `now` tables store data on current system metrics such as active queries.
- `history` tables store data historical metrics.
- `tail` tables are for data in transition. These tables are for internal use only and should not be queried by end users.

The database contains the following categories of tables:

- The database_* tables store query workload information for a Greenplum Database instance.
- The emcconnect_history table displays information about ConnectEMC events and alerts. ConnectEMC events are triggered based on a hardware failure, a fix to a failed hardware component, or a Greenplum Database startup. Once an ConnectEMC event is triggered, an alert is sent to EMC Support.
- The diskspace_* tables store diskspace metrics.
- The filerep_* tables store health and status metrics for the file replication process. This process is how high-availability/mirroring is achieved in Greenplum Database instance. Statistics are maintained for each primary-mirror pair.
- The health_* tables store system health metrics for the EMC Data Computing Appliance.
- The interface_stats_* tables store statistical metrics for each active interface of a Greenplum Database instance. Note: These tables are in place for future use and are not currently populated.
- The iterators_* tables store information about query plan iterators and their metrics. A query iterator refers to a node or operation in a query plan.
- The log_alert_* tables store information about pg_log errors and warnings.
- The queries_* tables store high-level query status information.
- The segment_* tables store memory allocation statistics for the Greenplum Database segment instances.
- The socket_stats_* tables store statistical metrics about socket usage for a Greenplum Database instance. Note: These tables are in place for future use and are not currently populated.
- The system_* tables store system utilization metrics.
- The tcp_stats_* tables store statistical metrics about TCP communications for a Greenplum Database instance. Note: These tables are in place for future use and are not currently populated.
- The udp_stats_* tables store statistical metrics about UDP communications for a Greenplum Database instance. Note: These tables are in place for future use and are not currently populated.

The Command Center database also contains the following views:

- The dynamic_memory_info view shows an aggregate of all the segments per host and the amount of dynamic memory used per host.
- The iterators_*_rollup set of views summarize the query iterator metrics across all segments in the system.
- The memory_info view shows per-host memory information from the `system_history` and `segment_history` tables.

database_*

The database_* tables store query workload information for a Greenplum Database instance. There are three database tables, all having the same columns:

- database_now is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. Current query workload data is stored in database_now during the period between data collection from the Command Center agents and automatic commitment to the database_history table.
- database_tail is an external table whose data files are stored in $MASTER_DATA_DIRECTORY/gpperfmon/data. This is a transitional table for query workload data that has been cleared from database_now but has not yet been committed to database_history. It typically only contains a few minutes worth of data.
- database_history is a regular table that stores historical database-wide query workload data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Column	Type	Description
ctime	timestamp	Time this row was created.
queries_total	int	The total number of queries in Greenplum Database at data collection time.
queries_running	int	The number of active queries running at data collection time.
queries_queued	int	The number of queries waiting in a resource queue at data collection time.

emcconnect_history

The `emcconnect_history` table displays information about ConnectEMC events and alerts. ConnectEMC events are triggered based on a hardware failure, a fix to a failed hardware component, or a Greenplum Database instance startup. Once an ConnectEMC event is triggered, an alert is sent to EMC Support.

This table is pre-partitioned into monthly partitions. Partitions are automatically added in one month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Note: This table only applies to Greenplum Data Computing Appliance platforms.

`ctime`	timestamp(0) without time zone	Time this ConnectEMC event occurred.
`hostname`	varchar(64)	The hostname associated with the ConnectEMC event.
`symptom_code`	int	A general symptom code for this type of event. For a list of symptom codes, see the *EMC Greenplum DCA Installation and Configuration Guide*.
`detailed_symptom_code`	int	A specific symptom code for this type of event.
`description`	text	A description of this type of event, based on the `detailed_symptom_code`.
`snmp_oid`	text	The SNMP object ID of the element/component where the event occurred, where applicable.
`severity`	text	The severity level of the ConnectEMC event. One of: `WARNING`: A condition that might require immediate attention. `ERROR`: An error occurred on the DCA. System operation and/or performance is likely affected. This alert requires immediate attention. `UNKNOWN`: This severity level is associated with hosts and devices on the DCA that are either disabled (due to hardware failure) or unreachable for some other reason. This alert requires immediate attention. `INFO`: A previously reported error condition is now resolved. Greenplum Database startup also triggers an INFO alert.
`status`	text	The current status of the system. The status is always `OK` unless a connection to the server/switch cannot be made, in which case the status is `FAILED`.
`attempted_transport`	boolean	`True` if an attempt was made to send an alert to EMC support. `False` if your system was configured not to send alerts.
`message`	text	The text of the error message created as a result of this event.

diskspace_*

The `diskspace_*` tables store diskspace metrics.

- `diskspace_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current diskspace metrics are stored in `database_now` during the period between data collection from the Command Center agents and automatic commitment to the `diskspace_history` table.
- `diskspace_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for diskspace metrics that have been cleared from `diskspace_now` but has not yet been committed to `diskspace_history`. It typically only contains a few minutes worth of data.
- `diskspace_history` is a regular table that stores historical diskspace metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Column	Type	Description
`ctime`	timestamp(0) without time zone	Time of diskspace measurement.
`hostname`	varchar(64)	The hostname associated with the diskspace measurement.
`Filesystem`	text	Name of the filesystem for the diskspace measurement.
`total_bytes`	bigint	Total bytes in the file system.
`bytes_used`	bigint	Total bytes used in the file system.
`bytes_available`	bigint	Total bytes available in file system.

filerep_*

The `filerep*` tables store high-availability file replication process information for a Greenplum Database instance. There are three filerep tables, all having the same columns:

- `filerep_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current file replication data is stored in `filerep_now` during the period between data collection from the Command Center agents and automatic commitment to the `filerep_history` table.
- `filerep_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for file replication data that has been cleared from `filerep_now` but has not yet been committed to `filerep_history`. It typically only contains a few minutes worth of data.
- `filerep_history` is a regular table that stores historical database-wide file replication data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

`ctime`	timestamp	Time this row was created.
`primary_measurement_microsec`	bigint	The length of time over which primary metrics (contained in UDP messages) were gathered.
`mirror_measurement_microsec`	bigint	The length of time over which mirror metrics (contained in UDP messages) were gathered.
`primary_hostname`	varchar(64)	The name of the primary host.
`primary_port`	int	The port number of the primary host.
`mirror_hostname`	varchar(64)	The name of the mirror host.
`mirror_port`	int	The port number of the mirror host.
`primary_write_syscall_bytes_avg`	bigint	The average amount of data written to disk on the primary for write system calls per interval.
`primary_write_syscall_byte_max`	bigint	The maximum amount of data written to disk on the primary for write system calls per interval.
`primary_write_syscall_microsecs_avg`	bigint	The average time required for a write system call to write data to disk on the primary per interval.
`primary_write_syscall_microsecs_max`	bigint	The maximum time required for a write system call to write data to disk on the primary per interval.
`primary_write_syscall_per_sec`	double precision	The number of write system calls on the primary per second. It reflects only the time to queue the write to disk in memory.
`primary_fsync_syscall_microsec_avg`	bigint	The average amount of time required for a file sync system call to write data to disk on the primary per interval.
`primary_fsync_syscall_microsec_max`	bigint	The maximum amount of time required for a file sync system call to write data to disk on the primary per interval.
`primary_fsync_syscall_per_sec`	double precision	The number of file sync system calls on the primary per second. Unlike write system calls which return immediately after the data is posted/queued, file sync system calls wait for all outstanding writes to be written to disk. File sync system call values in this column reflect actual disk access times for potentially large amounts of data.
[illegible]	bigint	The average amount of data written to shared

`primary_write_shmem_bytes_avg`	bigint	memory on the primary per interval.
`primary_write_shmem_bytes_max`	bigint	The maximum amount of data written to shared memory on the primary per interval.
`primary_write_shmem_microsec_avg`	bigint	The average amount of time required to write data to shared memory on the primary per interval.
`primary_write_shmem_microsec_max`	bigint	The maximum amount of time required to write data to shared memory on the primary per interval.
`primary_write_shmem_per_sec`	double precision	The number of writes to shared memory on the primary per second.
`primary_fsync_shmem_microsec_avg`	bigint	The average amount of time required by the file sync system call to write data to shared memory on the primary per interval.
`primary_fsync_shmem_microsec_max`	bigint	The maximum amount of time required by the file sync system call to write data to shared memory on the primary per interval.
`primary_fsync_shmem_per_sec`	double precision	The number of file sync calls to shared memory on the primary per second. File sync system call values in this column reflect actual disk access times for potentially large amounts of data.
`primary_write_shmem_per_sec`	double precision	The number of writes to shared memory on the primary per second.
`primary_fsync_shmem_microsec_avg`	bigint	The average amount of time required by the file sync system call to write data to shared memory on the primary per interval.
`primary_fsync_shmem_microsec_max`	bigint	The maximum amount of time required by the file sync system call to write data to shared memory on the primary per interval.
`primary_fsync_shmem_per_sec`	double precision	The number of file sync calls to shared memory on the primary per second. File sync system call values in this column reflect actual disk access times for potentially large amounts of data.
`primary_roundtrip_fsync_msg_microsec_avg`	bigint	The average amount of time required for a roundtrip file sync between the primary and the mirror per interval. This includes: 1. The queuing of a file sync message from the primary to the mirror. 2. The message traversing the network. 3. The execution of the file sync by the mirror. 4. The file sync acknowledgement traversing the network back to the primary.
		The maximum amount of time required for a roundtrip file sync between the primary and the mirror per interval. This includes:

`primary_roundtrip_fsync_msg_microsec_max`	bigint	1. The queuing of a file sync message from the primary to the mirror. 2. The message traversing the network. 3. The execution of the file sync by the mirror. 4. The file sync acknowledgement traversing the network back to the primary.
`primary_roundtrip_fsync_msg_per_sec`	double precision	The number of roundtrip file syncs per second.
`primary_roundtrip_test_msg_microsec_avg`	bigint	The average amount of time required for a roundtrip test message between the primary and the mirror to complete per interval. This is similar to `primary_roundtrip_fsync_msg_microsec_avg`, except it does not include a disk access component. Because of this, this is a useful metric that shows the average amount of network delay in the file replication process.
`primary_roundtrip_test_msg_microsec_max`	bigint	The maximum amount of time required for a roundtrip test message between the primary and the mirror to complete per interval. This is similar to `primary_roundtrip_fsync_msg_microsec_max`, except it does not include a disk access component. Because of this, this is a useful metric that shows the maximum amount of network delay in the file replication process.
`primary_roundtrip_test_msg_per_sec`	double precision	The number of roundtrip file syncs per second. This is similar to `primary_roundtrip_fsync_msg_per_sec`, except it does not include a disk access component. As such, this is a useful metric that shows the amount of network delay in the file replication process. Note that test messages typically occur once per minute, so it is common to see a value of "0" for time periods not containing a test message.
`mirror_write_syscall_size_avg`	bigint	The average amount of data written to disk on the mirror for write system calls per interval.
`mirror_write_syscall_size_max`	bigint	The maximum amount of data written to disk on the mirror for write system calls per interval.
`mirror_write_syscall_microsec_avg`	bigint	The average time required for a write system call to write data to disk on the mirror per interval.
`mirror_write_syscall_microsec_max`	bigint	The maximum time required for a write system call to write data to disk on the mirror per interval.
`primary_roundtrip_test_msg_per_sec`	double precision	The number of roundtrip file syncs per second. This is similar to `primary_roundtrip_fsync_msg_per_sec`, except it does not include a disk access component. As such, this is a useful metric that shows the amount of network delay in the file replication process.

		Note that test messages typically occur once per minute, so it is common to see a value of “0” for time periods not containing a test message.
`mirror_write_syscall_size_avg`	bigint	The average amount of data written to disk on the mirror for write system calls per interval.
`mirror_write_syscall_size_max`	bigint	The maximum amount of data written to disk on the mirror for write system calls per interval.
`mirror_write_syscall_microsec_avg`	bigint	The average time required for a write system call to write data to disk on the mirror per interval.

health_*

The `health_*` tables store system health metrics for the EMC Data Computing Appliance. There are three health tables, all having the same columns:

Note: This table only applies to Greenplum Data Computing Appliance platforms.

- `health_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current system health data is stored in `system_now` during the period between data collection from the Command Center agents and automatic commitment to the `system_history` table.
- `health_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for system health data that has been cleared from `system_now` but has not yet been committed to `system_history`. It typically only contains a few minutes worth of data.
- `health_history` is a regular table that stores historical system health metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Column	Type	Description
ctime	timestamp(0) without time zone	Time this snapshot of health information about this system was created.
hostname	varchar(64)	Segment or master hostname associated with this health information.
`symptom_code`	int	The symptom code related to the current health/status of an element or component of the system.
`detailed_symptom_code`	int	A more granular symptom code related to the health/status of a element or component of the system.
`description`	text	A description of the health/status of this symptom code.
`snmp_oid`	text	The SNMP object ID of the element/component where the event occurred, where applicable.
`status`	text	The current status of the system. The status is always `OK` unless a connection to the server/switch cannot be made, in which case the status is `FAILED`.
`message`	text	The text of the error message created as a result of this event.

interface_stats_*

The `interface_stats_*` tables store statistical metrics about communications over each active interface for a Greenplum Database instance.

These tables are in place for future use and are not currently populated.

There are three `interface_stats` tables, all having the same columns:

- `interface_stats_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`.
- `interface_stats_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for statistical interface metrics that has been cleared from `interface_stats_now` but has not yet been committed to `interface_stats_history`. It typically only contains a few minutes worth of data.
- `interface_stats_history` is a regular table that stores statistical interface metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in one month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Column	Type	Description
`interface_name`	string	Name of the interface. For example: eth0, eth1, lo.
`bytes_received`	bigint	Amount of data received in bytes.
`packets_received`	bigint	Number of packets received.
`receive_errors`	bigint	Number of errors encountered while data was being received.
`receive_drops`	bigint	Number of times packets were dropped while data was being received.
`receive_fifo_errors`	bigint	Number of times FIFO (first in first out) errors were encountered while data was being received.
`receive_frame_errors`	bigint	Number of frame errors while data was being received.
`receive_compressed_packets`	int	Number of packets received in compressed format.
`receive_multicast_packets`	int	Number of multicast packets received.
`bytes_transmitted`	bigint	Amount of data transmitted in bytes.
`packets_transmitted`	bigint	Amount of data transmitted in bytes.
`packets_transmitted`	bigint	Number of packets transmitted.
`transmit_errors`	bigint	Number of errors encountered during data transmission.
`transmit_drops`	bigint	Number of times packets were dropped during data transmission.
`transmit_fifo_errors`	bigint	Number of times fifo errors were encountered during data transmission.
`transmit_collision_errors`	bigint	Number of times collision errors were encountered during data transmission.
`transmit_carrier_errors`	bigint	Number of times carrier errors were encountered during data transmission.
`transmit_compressed_packets`	int	Number of packets transmitted in compressed format.

iterators_*

The `iterators_*` tables store information about query plan iterators and their metrics. A query iterator refers to a node or operation in a query plan. For example, a sequential scan operation on a table may be one type of iterator in a particular query plan.

The `tmid`, `ssid` and `ccnt` columns are the composite key that uniquely identifies a particular query. These columns can be used to join with the `queries_*` data tables.

There are three iterator tables, all having the same columns:

- `iterators_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current query plan iterator data is stored in `iterators_now` during the period between data collection from the Command Center agents and automatic commitment to the `iterators_history` table.
- `iterators_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for query plan iterator data that has been cleared from `iterators_now` but has not yet been committed to `iterators_history`. It typically only contains a few minutes worth of data.
- `iterators_history` is a regular table that stores historical query plan iterator data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

See also the iterator_rollup views for summary metrics of the query plan iterator data.

`ctime`	timestamp	Time this row was created.
`tmid`	int	A time identifier for a particular query. All iterator records associated with the query will have the same tmid.
`ssid`	int	The session id as shown by the `gp_session_id` parameter. All iterator records associated with the query will have the same `ssid`.
`ccnt`	int	The command number within this session as shown by `gp_command_count` parameter. All iterator records associated with the query will have the same `ccnt`.
`segid`	int	The segment ID (`dbid` from `gp_segment_configuration`).
`pid`	int	The postgres process ID for this iterator.
`nid`	int	The query plan node ID from the Greenplum slice plan.
`pnid`	int	The parent query plan node ID from the Greenplum slice plan.
`hostname`	varchar(64)	Segment hostname.
`ntype`	varchar(64)	The iterator operation type. Possible values are listed in [Iterator Metrics](db-iterator-metrics.html).
`nstatus`	varchar(64)	The status of this iterator. Possible values are: Initialize, Executing and Finished.
`tstart`	timestamp	Start time for the iterator.
`tduration`	int	Duration of the execution.
`pmemsize`	bigint	Maximum work memory allocated by the Greenplum planner to this iterator's query process.
`memsize`	bigint	OS memory allocated to this iterator's process.
`memresid`	bigint	Resident memory allocated to this iterator's process (as opposed to shared memory).

`memshare`	bigint	Shared memory allocated to this iterator's process.
`cpu_elapsed`	bigint	Total CPU usage of the process executing the iterator.
`cpu_currpct`	float	The percentage of CPU currently being utilized by this iterator process. This value is always zero for historical (completed) iterators.
`rowsout`	bigint	The actual number of rows output by the iterator.
`rowsout_est`	bigint	The query planner's estimate of rows output by the iterator.
`m0_name`	varchar(64)	Each operation in a query plan (ntype) has metrics associated with it. For all operations, this metric name is `Rows In` .
`m0_unit`	varchar(64)	The unit of measure for this metric. For all operations (`ntype`), this unit of measure is `Rows` .
`m0_val`	bigint	The value of this metric.
`m0_est`	bigint	The estimated value of this metric.
`m1_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m1_unit`	varchar(64)	The unit of measure for this metric. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m1_val`	bigint	The value of this metric.
`m1_est`	bigint	The estimated value of this metric.
`m2_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m2_unit`	varchar(64)	The unit of measure for this metric. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m2_val`	bigint	The value of this metric.
`m2_est`	bigint	The estimated value of this metric.
`m3_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m3_unit`	varchar(64)	The unit of measure for this metric. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m3_val`	bigint	The value of this metric.
`m3_est`	bigint	The estimated value of this metric.
`m4_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of

		iterator attributes and their corresponding units.
`m4_unit`	varchar(64)	The unit of measure for this metric. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m4_val`	bigint	The value of this metric.
`m4_est`	bigint	The estimated value of this metric.
`m5_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m5_unit`	varchar(64)	The unit of measure for this metric. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m5_val`	bigint	The value of this metric.
`m5_est`	bigint	The estimated value of this metric.
`m6_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m6_unit`	varchar(64)	The unit of measure for this metric. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m6_val`	bigint	The value of this metric.
`m6_est`	bigint	The estimated value of this metric.
`m7_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m7_unit`	varchar(64)	The unit of measure for this metric. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m7_val`	bigint	The value of this metric.
`m7_est`	bigint	The estimated value of this metric.
`m8_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m8_unit`	varchar(64)	The unit of measure for this metric. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m8_val`	bigint	The actual value of this metric.
`m8_est`	bigint	The estimated value of this metric.
`m9_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See [Iterator Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
		The unit of measure for this metric. See [Iterator

`m9_unit`	varchar(64)	Metrics](db-iterator-metrics.html) for a complete list of iterator attributes and their corresponding units.
`m9_val`	bigint	The actual value of this metric.
`m9_est`	bigint	The estimated value of this metric.
`m10_name` - `m15_name`	varchar(64)	The iterator name (`ntype`) associated with this metric. Metrics `m10` through `m15` are currently not used.
`m10_unit` - `m15_unit`	varchar(64)	The unit of measure for this metric. Metrics `m10` through `m15` are currently not used.
`m10_value` - `m15_value`	bigint	The actual value of this metric. Metrics `m10` through `m15` are currently not used.
`m10_est` - `m15_est`	bigint	The estimated value of this metric. Metrics `m10` through `m15` are currently not used.
`t0_name`	varchar(64)	This column is a label for `t0_val`. Its value is always `Name`.
`t0_val`	varchar(128)	The name of the relation being scanned by an iterator. This metric is collected only for iterators that perform scan operations such as a sequential scan or function scan.

Iterator Metrics

The tables in this section list all possible iterators in a query on Greenplum Database instance. The iterator tables include the metric name, the column in the `iterators_*` table in the `gpperfmon` database where the metric appears, how the metric is measured (unit), and a description of the metric.

Metric Terminology

The following information explains some of the database terms and concepts that appear in iterator metrics in Greenplum Database:

Node
Refers to a step in a query plan. A query plan has sets of operations that Greenplum Database performs to produce the answer to a given query. A node in the plan represents a specific database operation, such as a table scan, join, aggregation, sort, etc.

Iterator
Represents the actual execution of the node in a query plan. Node and iterator are sometimes used interchangeably.

Tuple
Refers to a row returned as part of a result set from a query, as well as a record in a table.

Spill
When there is not enough memory to perform a database operation, data must be written (or spilled) to disk.

Passes
Occur when an iterator must scan (or pass) over spilled data to obtain a result. A pass represents one pass through all input tuples, or all data in batch files generated after spill, which happens hierarchically. In the first pass, all input tuples are read, and intermediate results are spilled to a specified number of batch files. In the second pass, the data in all batch files is processed. If the results are still too large to store in memory, the intermediate results are spilled to the second level of spill files, and the process repeats again.

Batches
Refers to the actual files created when data is spilled to disk. This is most often associated to Hash operations.

Join
This clause in a query joins two or more tables. There are three types of Join algorithms in Greenplum Database instance:

- Hash Join
- Merge Join
- Nested Loop

Each of these operations include their own respective Join semantics. The Command Center Console displays iterator metrics for each of these semantics.

Append

An Append iterator has two or more input sets. Append returns all rows from the first input set, then all rows from the second input set, and so on, until all rows from all input sets are processed. Append is also used when you select from a table involved in an inheritance hierarchy.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Append Current Input Source	`m1_name`	Inputs	The number of the current table being scanned.

Append-Only Scan

This iterator scans append-only type-tables.

Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Append-only Scan Rescan	`m1_name`	Rescans	The number of append-only rescans by this iterator.

Append-only Columnar Scan

This iterator scans append-only columnar-type tables.

Rows in	`m0_name`	Rows	The number of tuples received by

			the iterator.
Append-Only Columnar Scan Rescan	`m1_name`	Rescans	The number of append-only columnar rescans by this iterator.

Aggregate

The query planner produces an aggregate iterator whenever the query includes an aggregate function. For example, the following functions are aggregate functions: `AVG()`, `COUNT()`, `MAX()`, `MIN()`, `STDDEV()`, `SUM()`, and `VARIANCE()`. Aggregate reads all the rows in the input set and computes the aggregate values. If the input set is not grouped, Aggregate produces a single result row.

Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Aggregate Total Spill Tuple	`m1_name`	Tuples	The number of tuples spilled to disk
Aggregate Total Spill Bytes	`m2_name`	Bytes	The number of bytes spilled to disk.
Aggregate Total Spill Batches	`m3_name`	Batches	The number of spill batches required.
Aggregate Total Spill Pass	`m4_name`	Passes	The number of passes across all of the batches.
Aggregate Current Spill Pass Read Tuples	`m5_name`	Tuples	The number of bytes read in for this spill batch.
Aggregate Current Spill Pass Read Bytes	`m6_name`	Bytes	The number of tuples read in for this spill batch.
Aggregate Current Spill Pass Tuples	`m7_name`	Tuples	The number of tuples that are in each spill file in the current pass.
Aggregate Current Spill Pass Bytes	`m8_name`	Bytes	The number of bytes that are in each spill file in the current pass.
Aggregate Current Spill Pass Batches	`m9_name`	Batches	The number of batches created in the current pass.

BitmapAnd

This iterator takes the bitmaps generated from multiple BitmapIndexScan iterators, puts them together with an `AND` clause, and generates a new bitmap as its output.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

BitmapOr

This iterator takes the bitmaps generated from multiple BitmapIndexScan iterators, puts them together with an `OR` clause, and generates a new bitmap as its output.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

Bitmap Append-Only Scan

This iterator retrieves all rows from the bitmap generated by BitmapAnd, BitmapOr, or BitmapIndexScan and accesses the append-only table to retrieve the relevant rows.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

Bitmap Heap Scan

This iterator retrieves all rows from the bitmap generated by BitmapAnd, BitmapOr, or BitmapIndexScan and accesses the heap table to retrieve the relevant rows.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Bitmap Heap Scan Pages	`m1_name`	Pages	The number of bitmap heap pages scanned.
Bitmap Heap Scan Rescan	`m2_name`	Rescans	The number of bitmap heap page rescans by this iterator.

Bitmap Index Scan

This iterator produces a bitmap that corresponds to the rules that satisfy the query plan.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Bitmap Index Scan Rescan	`m1_name`	Rescans	The number of bitmap index rescans by this iterator.

Broadcast Motion

Note that the `Motion` metrics for the iterator are useful when investigating potential networking issues in the Greenplum Database system. Typically, the "Ack Time" values should be very small (microseconds or milliseconds). However if the "Ack Time" values are one or more seconds (particularly the "Motion Min Ack Time" metric), then a network performance issue likely exists.

Also, if there are a large number of packets being dropped because of queue overflow, you can increase the value for the `gp_interconnect_queue_depth` system configuration parameter to improve performance. See the *Greenplum Database Reference Guide*for more in formation about system configuration parameters.

Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Motion Bytes Sent	`m1_name`	Bytes	The number of bytes sent by the iterator.
Motion Total Ack Time	`m2_name`	Microseconds	The total amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Average Ack Time	`m3_name`	Microseconds	The average amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Max Ack Time	`m4_name`	Microseconds	The maximum amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Min Ack Time	`m5_name`	Microseconds	The minimum amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Count Resent	`m6_name`	Packets	The total number of packets that the iterator did not acknowledge when they first arrived in the queue.
Motion Max Resent	`m7_name`	Packets	The maximum number of packets that the iterator did not acknowledge when they first arrived in the queue. This metric is applied on a per packet basis. For example, a value of "10" indicates that a particular packet did not get acknowledged by this iterator 10 times, and that this was the maximum for this iterator.
Motion Bytes Received	`m8_name`	Bytes	The number of bytes received by the iterator.

Motion Count Dropped	`m9_name`	Packets	The number of packets dropped by the iterator because of buffer overruns.

Explicit Redistribute Motion

The Explicit Redistribute iterator moves tuples to segments explicitly specified in the segment ID column of the tuples. This differs from a Redistribute Motion iterator, where target segments are indirectly specified through hash expressions. The Explicit Redistribute iterator is used when the query portion of a DML planned statement requires moving tuples across distributed tables.

Note that the Motion metrics for the iterator are useful when investigating potential networking issues in the Greenplum Database system. Typically, the "Ack Time" values should be very small (microseconds or milliseconds). However if the "Ack Time" values are one or more seconds (particularly the "Motion Min Ack Time" metric), then a network performance issue likely exists.

Also, if there are a large number of packets being dropped because of queue overflow, you can increase the value for the `gp_interconnect_queue_depth` system configuration parameter to improve performance. See the *Greenplum Database Reference Guide*for more in formation about system configuration parameters.

.

Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Motion Bytes Sent	`m1_name`	Bytes	The number of bytes sent by the iterator.
Motion Total Ack Time	`m2_name`	Microseconds	The total amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Average Ack Time	`m3_name`	Microseconds	The average amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Max Ack Time	`m4_name`	Microseconds	The maximum amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Min Ack Time	`m5_name`	Microseconds	The minimum amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Count Resent	`m6_name`	Packets	The total number of packets that the iterator did not acknowledge when they first arrived in the queue.
Motion Max Resent	`m7_name`	Packets	The maximum number of packets that the iterator did not acknowledge when they first arrived in the queue. This metric is applied on a per packet basis. For example, a value of "10" indicates that a particular packet did not get acknowledged by this iterator 10 times, and that this was the maximum for this iterator.
Motion Bytes Received	`m8_name`	Bytes	The number of bytes received by the iterator.
Motion Count Dropped	`m9_name`	Packets	The number of packets dropped by the iterator because of buffer overruns.

External Scan

This iterator scans an external table.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
External Scan Rescan	`m1_name`	Rescans	The number of external table rescans by this iterator.

Function Scan

This iterator returns tuples produced by a function.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

Gather Motion

This iterator gathers streams of tuples that are sent by "sending" motions. If a merge key is specified, it merges many streams into a single order-preserved stream.

Note that the Motion metrics for the iterator are useful when investigating potential networking issues in the Greenplum Database system. Typically, the "Ack Time" values should be very small (microseconds or milliseconds). However if the "Ack Time" values are one or more seconds (particularly the "Motion Min Ack Time" metric), then a network performance issue likely exists.

Also, if there are a large number of packets being dropped because of queue overflow, you can increase the value for the `gp_interconnect_queue_depth` system configuration parameter to improve performance. See the *Greenplum Database Reference Guide*for more in formation about system configuration parameters.

Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Motion Bytes Sent	`m1_name`	Bytes	The number of bytes sent by the iterator.
Motion Total Ack Time	`m2_name`	Microseconds	The total amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Average Ack Time	`m3_name`	Microseconds	The average amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Max Ack Time	`m4_name`	Microseconds	The maximum amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Min Ack Time	`m5_name`	Microseconds	The minimum amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Count Resent	`m6_name`	Packets	The total number of packets that the iterator did not acknowledge when they first arrived in the queue.
Motion Max Resent	`m7_name`	Packets	The maximum number of packets that the iterator did not acknowledge when they first arrived in the queue. This metric is applied on a per packet basis. For example, a value of "10" indicates that a particular packet did not get acknowledged by this iterator 10 times, and that this was the maximum for this iterator.

Motion Bytes Received	`m8_name`	Bytes	The number of bytes received by the iterator.
Motion Count Dropped	`m9_name`	Packets	The number of packets dropped by the iterator because of buffer overruns.

Group Aggregate

The GroupAggregate iterator is a way to compute vector aggregates, and it is used to satisfy a `GROUP BY` clause. A single input set is required by the GroupAggregate iterator, and it must be ordered by the grouping column(s). This iterator returns a single row for a unique value of grouping columns.

Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Aggregate Total Spill Tuples	`m1_name`	Tuples	The number of tuples spilled to disk.
Aggregate Total Spill Bytes	`m2_name`	Bytes	The number of bytes spilled to disk.
Aggregate Total Spill Batches	`m3_name`	Batches	The number of spill batches required.
Aggregate Total Spill Pass	`m4_name`	Passes	The number of passes across all of the batches.
Aggregate Current Spill Pass Read Tuples	`m5_name`	Tuples	The number of bytes read in for this spill batch
Aggregate Current Spill Pass Read Bytes	`m6_name`	Bytes	The number of tuples read in for this spill batch
Aggregate Current Spill Pass Tuples	`m7_name`	Tuples	The number of tuples that are in each spill file in the current pass.
Aggregate Current Spill Pass Bytes	`m8_name`	Bytes	The number of bytes that are in each spill file in the current pass.
Aggregate Current Spill Pass Batches	`m9_name`	Batches	The number of batches created in the current pass.

Hash Join

The Hash Join iterator requires two input sets - the outer and inner tables.

The Hash Join iterator starts by creating its inner table using the Hash operator. The Hash operator creates a temporary Hash index that covers the join column in the inner table. When the hash table (that is, the inner table) is created, Hash Join reads each row in the outer table, hashes the join column (from the outer table), and searches the temporary Hash index for a matching value.

In a Greenplum Database instance, a Hash Join algorithm can be used with the following join semantics:

- Left Join
- Left Anti Semi Join
- Full Join
- Right Join
- EXISTS Join
- Reverse In Join
- Unique Inner Join
- Unique Outer Join

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Hash Spill Batches	`m1_name`	Batches	The current batch being spilled.
Hash Spill Tuples	`m2_name`	Tuples	The current number of spilled tuples.
Hash Spill Bytes	`m3_name`	Bytes	The current number of bytes spilled to disk.

HashAggregate

The HashAggregate iterator is similar to the GroupAggregate iterator. A single input set is required by the HashAggregate iterator and it creates a hash table from the input. However, it does not require its input to be ordered.

Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Aggregate Total Spill Tuples	`m1_name`	Tuples	The number of tuples spilled to disk.
Aggregate Total Spill Bytes	`m2_name`	Bytes	The number of bytes spilled to disk.
Aggregate Total Spill Batches	`m3_name`	Batches	The number of spill batches required.
Aggregate Total Spill Pass	`m4_name`	Passes	The number of passes across all of the batches.
Aggregate Current Spill Pass Read Tuples	`m5_name`	Tuples	The number of bytes read in for this spill batch
Aggregate Current Spill Pass Read Bytes	`m6_name`	Bytes	The number of tuples read in for this spill batch
Aggregate Current Spill Pass Tuples	`m7_name`	Tuples	The number of tuples that are in each spill file in the current pass.
Aggregate Current Spill Pass Bytes	`m8_name`	Bytes	The number of bytes that are in each spill file in the current pass.
Aggregate Current Spill Pass Batches	`m9_name`	Batches	The number of batches created in the current pass.

Index Scan

An Index Scan operator traverses an index structure. If you specify a starting value for an indexed column, the Index Scan will begin at the appropriate value. If you specify an ending value, the Index Scan will complete as soon as it finds an index entry greater than the ending value. A query planner uses an Index Scan operator when it can reduce the size of the result set by traversing a range of indexed values, or when it can avoid a sort because of the implicit ordering offered by an index.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Index Scan Restore	`m1_name`	Restores	The number of restores.
Index Scan Rescan	`m2_name`	Rescans	The number of rescans.

Limit

The Limit operator is used to limit the size of a result set. A Greenplum Database instance uses the Limit operator for both Limit and Offset processing. The Limit operator works by discarding the first x rows from its input set, returning the next y rows, and discarding the remainder. If the query includes an OFFSET clause, x represents the offset amount; otherwise, x is zero. If the query includes a LIMIT clause, y represents the Limit amount; otherwise, y is at least as large as the number of rows in the input set.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

Materialize

The materialize iterator is used for some sub-select operations. The query planner can decide that it is less expensive to materialize a sub-select one time than it is to repeat the work for each top-level row. Materialize is also used for some merge/join operations.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Materialize Rescan	`m1_name`	Rescans	The number of times the executor requested to rescan the date for this iterator.

Merge Join

The Merge Join iterator joins two tables. Like the Nested Loop iterator, Merge Join requires two input sets: An outer table and an inner table. Each input set must be ordered by the join columns. In a Greenplum Database instance, the Merge Join algorithm can be used with the following join semantics:

- Left Join
- Left Anti Semi Join
- Full Join
- Right Join
- EXISTS Join
- Reverse In Join
- Unique Outer joins
- Unique Inner Join

Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Merge Join Inner Tuples	`m1_name`	Tuples	The number of rows from the inner part of the query plan.
Merge Join Outer Tuples	`m2_name`	Tuples	The number of rows from the Outer part of the query plan.

Nested Loop

The Nested Loop iterator is used to perform a join between two tables, and as a result requires two input sets. It fetches each table from one of the input sets (called the outer table). For each row in the outer table, the other input (called the inner table) is searched for a row that meets the join qualifier. In a Greenplum Database instance, a Merge Join algorithm can be used with the following join semantics:

- Left Join
- Left Anti Semi Join
- Full Join
- Right Join
- EXISTS Join
- Reverse In Join
- Unique Outer Join
- Unique Inner Join

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Nested Loop Inner Tuples	`m1_name`	Tuples	The number of rows from the inner part of the query plan.
Nested Loop Outer Tuples	`m2_name`	Tuples	The number of rows from the outer part of the query plan.

Redistribute Motion

This iterator sends an outbound tuple to only one destination based on the value of a hash.

Note that the Motion metrics for the iterator are useful when investigating potential networking issues in the Greenplum Database system. Typically, the "Ack Time" values should be very small (microseconds or milliseconds). However if the "Ack Time" values are one or more seconds (particularly the "Motion Min Ack Time" metric), then a network performance issue likely exists.

Also, if there are a large number of packets being dropped because of queue overflow, you can increase the value for the `gp_interconnect_queue_depth` system configuration parameter to improve performance. See the *Greenplum Database Reference Guide*for more in formation about system configuration parameters.

Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Motion Bytes Sent	`m1_name`	Bytes	The number of bytes sent by the iterator.
Motion Total Ack Time	`m2_name`	Microseconds	The total amount of time that the iterator waited for an acknowledgement after sending a packet of data.

Motion Average Ack Time	`m3_name`	Microseconds	The average amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Max Ack Time	`m4_name`	Microseconds	The maximum amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Min Ack Time	`m5_name`	Microseconds	The minimum amount of time that the iterator waited for an acknowledgement after sending a packet of data.
Motion Count Resent	`m6_name`	Packets	The total number of packets that the iterator did not acknowledge when they first arrived in the queue.
Motion Max Resent	`m7_name`	Packets	The maximum number of packets that the iterator did not acknowledge when they first arrived in the queue. This metric is applied on a per packet basis. For example, a value of "10" indicates that a particular packet did not get acknowledged by this iterator 10 times, and that this was the maximum for this iterator.
Motion Bytes Received	`m8_name`	Bytes	The number of bytes received by the iterator.
Motion Count Dropped	`m9_name`	Packets	The number of packets dropped by the iterator because of buffer overruns.

Result

The Result iterator is used to either (1) execute a query that does not retrieve data from a table, or evaluate the parts of a WHERE clause that do not depend on data retrieved from a table. It can also be used if the top node in the query plan is an Append iterator.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

Repeat

This iterator repeats every input operator a certain number of times. This is typically used for certain grouping operations.

Metric Description Rows in `m0_name` Rows The number of tuples received by the iterator.

Seq Scan

The Seq Scan iterator scans heap tables, and is the most basic query iterator. Any single-table query can be done by using the Seq Scan iterator. Seq Scan starts at the beginning of a heap table and scans to the end of the heap table. For each row in the heap table, Seq Scan evaluates the query constraints (the WHERE clause). If the constraints are satisfied, the required columns are added to the result set.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Seq Scan Page Stats	`m1_name`	Pages	The number of pages scanned.
Seq Scan Restore Pos	`m2_name`	Restores	The number of times the executor restored the scan position.
Seq Scan Rescan	`m3_name`	Rescans	The number of times the executor requested to rescan the date for this iterator.

SetOp

There are four SetOp iterators:

- Intersect
- Intersect All
- Except
- Except All

These iterators are produced only when the query planner encounters an `INTERSECT`, `INTERSECT ALL`, `EXCEPT`, or `EXCEPT ALL` clause, respectively.

All SetOp iterators require two input sets. They combine the input sets into a sorted list, and then groups of identical rows are identified. For each group, the SetOp iterators counts the number of rows contributed by each input set, then uses the counts to determine the number of rows to add to the result set.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

Shared Scan

This iterator evaluates the common parts of a query plan. It shares the output of the common sub-plans with all other iterators, so that the sub-plan only needs to execute one time.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Seq Scan Page Stats	`m1_name`	Pages	The number of pages scanned.
Seq Scan Restore Pos	`m2_name`	Restores	The number of times the executor restored the scan position.
Seq Scan Rescan	`m3_name`	Rescans	The number of times the executor requested to rescan the date for this iterator.

Sort

The Sort iterator imposes an ordering on the result set. A Greenplum Database instance uses two different sort strategies: An in-memory sort and an on-disk sort. If the size of the result set exceeds the available memory, the Sort iterator distributes the input set to a collection of sorted work files and then merges the work files back together again. If the result set is less than the available memory, the sort is done in memory.

The Sort iterator is used for many purposes. A Sort can be used to satisfy an `ORDER BY` clause. Also, some query operators require their input sets to be ordered.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Sort Memory Usage	`m1_name`	Bytes	The number of bytes used by the sort.
Sort Spill Tuples	`m2_name`	Tuples	The current number of spilled tuples.
Sort Spill Bytes	`m3_name`	Bytes	The current number of spilled bytes.
Sort Spill Pass	`m4_name`	Passes	The number of merge passes. Each merge pass merges several sorted runs into larger runs.
Sort Current Spill Pass Tuples	`m5_name`	Tuples	The number of tuples spilled in the current spill pass.
Sort Current Spill Pass Bytes	`m6_name`	Bytes	The number of bytes spilled in the current spill pass.

Subquery Scan

A Subquery Scan iterator is a pass-through iterator. It scans through its input set, adding each row to the result set. This iterator is used for internal purposes and has no affect on the overall query plan.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.
Subquery Scan Rescan	`m1_name`	Rescans	The number of times the executor requested to rescan the date for this iterator.

Tid Scan

The Tid Scan (tuple ID scan) iterator is used whenever the query planner encounters a constraint of the form `ctid = expression` or `expression = ctid`. This specifies a tuple ID, an identifier that is unique within a table. The tuple ID works like a bookmark, but is valid only within a single transaction. After the transaction completes, the tuple ID is not used again.

Metric	Metric Column	Unit	Description

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

Unique

The Unique iterator eliminates duplicate values from the input set. The input set must be ordered by the columns, and the columns must be unique. The Unique operator removes only rows — it does not remove columns and it does not change the ordering of the result set. Unique can return the first row in the result set before it has finished processing the input set. The query planner uses the Unique operator to satisfy a `DISTINCT` clause. Unique is also used to eliminate duplicates in a `UNION`.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

Values Scan

The Value Scan iterator is used to iterate over a set of constant tuples.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

Window

The Window function performs calculations across sets of rows that are related to the current query row. The Window iterator computes Window functions on the input set of rows.

Metric	Metric Column	Unit	Description
Rows in	`m0_name`	Rows	The number of tuples received by the iterator.

log_alert_*

The `log_alert_*` tables store `pg_log` errors and warnings.

There are three `log_alert` tables, each with the same columns:

- `log_alert_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current `pg_log` errors and warnings data is stored in `log_alert_now` during the period between data collection from the Command Center agents and commitment to the `log_alert_history` table.
- `log_alert_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for query workload data that has been cleared from `log_alert_now` but has not yet been committed to `log_alert_history`. It typically only contains a few minutes worth of data.
- `log_alert_history` is a regular table that stores historical database-wide errors and warnings data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Column	Type	Description
`logtime`	timestamp with time zone	Timestamp for this log
`loguser`	text	User of the query
`logdatabase`	text	The accessed database
`logpid`	text	Process id
`logthread`	text	Thread number
`loghost`	text	Host name or ip address
`logport`	text	Port number
`logsessiontime`	timestamp with time zone	Session timestamp
`logtransaction`	integer	Transaction id
`logsession`	text	Session id
`logcmdcount`	text	Command count
`logsegment`	text	Segment number
`logslice`	text	Slice number
`logdistxact`	text	Distributed transaction
`loglocalxact`	text	Local transaction
`logsubxact`	text	Subtransaction
`logseverity`	text	Log severity
`logstate`	text	State
`logmessage`	text	Log message
`logdetail`	text	Detailed message
`loghint`	text	Hint info
`logquery`	text	Executed query
`logquerypos`	text	Query position
`logcontext`	text	Context info
`logdebug`	text	Debug
`logcursorpos`	text	Cursor position
`logfunction`	text	Function info
`logfile`	text	Source code file
`logline`	text	Source code line
`logstack`	text	Stack trace

Log Processing and Rotation

The Greenplum Database system logger writes alert logs in the `$MASTER_DATA_DIRECTORY/gpperfmon/logs` directory.

The agent process (`gpmmon`) performs the following steps to consolidate log files and load them into the `gpperfmon` database:

1. Gathers all of the `gpdb-alert-*` files in the logs directory (except the latest, which the syslogger has open and is writing to) into a single file, `alert_log_stage`.
2. Loads the `alert_log_stage` file into the `log_alert_history` table in the `gpperfmon` database.
3. Truncates the `alert_log_stage` file.
4. Removes all of the `gp-alert-*` files, except the latest.

The syslogger rotates the alert log every 24 hours or when the current log file reaches or exceeds 1MB. A rotated log file can exceed 1MB if a single error message contains a large SQL statement or a large stack trace. Also, the syslogger processes error messages in chunks, with a separate chunk for each logging process. The size of a chunk is OS-dependent; on Red Hat Enterprise Linux, for example, it is 4096 bytes. If many Greenplum Database sessions generate error messages at the same time, the log file can grow significantly before its size is checked and log rotation is triggered.

queries_*

The `queries_*` tables store high-level query status information.

The `tmid` , `ssid` and `ccnt` columns are the composite key that uniquely identifies a particular query. These columns can be used to join with the `iterators_*` tables.

There are three queries tables, all having the same columns:

- `queries_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data` . Current query status is stored in `queries_now` during the period between data collection from the Command Center agents and automatic commitment to the `queries_history` table.
- `queries_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data` . This is a transitional table for query status data that has been cleared from `queries_now` but has not yet been committed to `queries_history` . It typically only contains a few minutes worth of data.
- `queries_history` is a regular table that stores historical query status data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

`ctime`	timestamp	Time this row was created.
`tmid`	int	A time identifier for a particular query. All records associated with the query will have the same `tmid` .
`ssid`	int	The session id as shown by `gp_session_id` . All records associated with the query will have the same `ssid` .
`ccnt`	int	The command number within this session as shown by `gp_command_count` . All records associated with the query will have the same `ccnt` .
`username`	varchar(64)	Greenplum role name that issued this query.
`db`	varchar(64)	Name of the database queried.
`cost`	int	Not implemented in this release.
`tsubmit`	timestamp	Time the query was submitted.
`tstart`	timestamp	Time the query was started.
`tfinish`	timestamp	Time the query finished.
`status`	varchar(64)	Status of the query – `start` , `done` , or `abort` .
`rows_out`	bigint	Rows out for the query.
`cpu_elapsed`	bigint	CPU usage by all processes across all segments executing this query (in seconds). It is the sum of the CPU usage values taken from all active primary segments in the database system. Note that Greenplum Command Center logs the value as 0 if the query runtime is shorter than the value for the quantum. This occurs even if the query runtime is greater than the values for `min_query_time` and `min_detailed_query` , and these values are lower than the value for the quantum.
`cpu_currpct`	float	Current CPU percent average for all processes executing this query. The percentages for all processes running on each segment are averaged, and then the average of all those values is calculated to render this metric.

		Current CPU percent average is always zero in historical and tail data.
`skew_cpu`	float	Displays the amount of processing skew in the system for this query. Processing/CPU skew occurs when one segment performs a disproportionate amount of processing for a query. This value is the coefficient of variation in the CPU% metric of all iterators across all segments for this query, multiplied by 100. For example, a value of .95 is shown as 95.
`skew_rows`	float	Displays the amount of row skew in the system. Row skew occurs when one segment produces a disproportionate number of rows for a query. This value is the coefficient of variation for the `rows_in` metric of all iterators across all segments for this query, multiplied by 100. For example, a value of .95 is shown as 95.
`query_hash`	bigint	Not implemented in this release.
`query_text`	text	The SQL text of this query.
`query_plan`	text	Text of the query plan. Not implemented in this release.
`application_name`	varchar(64)	The name of the application.
`rsqname`	varchar(64)	The name of the resource queue.
`rqppriority`	varchar(64)	The priority of the query – `max, high, med, low, or min`.

segment_*

The `segment_*` tables contain memory allocation statistics for the Greenplum Database segment instances. This tracks the amount of memory consumed by all postgres processes of a particular segment instance, and the remaining amount of memory available to a segment as per the setting of the `postgresql.conf` configuration parameter: `gp_vmem_protect_limit`. Query processes that cause a segment to exceed this limit will be cancelled in order to prevent system-level out-of-memory errors. See the *Greenplum Database Reference Guide*for more information about this parameter.

There are three segment tables, all having the same columns:

- `segment_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current memory allocation data is stored in `segment_now` during the period between data collection from the Command Center agents and automatic commitment to the segment_history table.
- `segment_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for memory allocation data that has been cleared from `segment_now` but has not yet been committed to `segment_history`. It typically only contains a few minutes worth of data.
- `segment_history` is a regular table that stores historical memory allocation metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

A particular segment instance is identified by its `hostname` and `dbid` (the unique segment identifier as per the `gp_segment_configuration` system catalog table).

`ctime`	timestamp(0) (without time zone)	The time the row was created.
`dbid`	int	The segment ID (`dbid` from `gp_segment_configuration`).
`hostname`	charvar(64)	The segment hostname.
`dynamic_memory_used`	bigint	The amount of dynamic memory (in bytes) allocated to query processes running on this segment.
`dynamic_memory_available`	bigint	The amount of additional dynamic memory (in bytes) that the segment can request before reaching the limit set by the `gp_vmem_protect_limit` parameter.

See also the views `memory_info` and `dynamic_memory_info` for aggregated memory allocation and utilization by host.

socket_stats_*

The `socket_stats_*` tables store statistical metrics about socket usage for a Greenplum Database instance. There are three system tables, all having the same columns:

These tables are in place for future use and are not currently populated.

- `socket_stats_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`.
- `socket_stats_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for socket statistical metrics that has been cleared from `socket_stats_now` but has not yet been committed to `socket_stats_history`. It typically only contains a few minutes worth of data.
- `socket_stats_history` is a regular table that stores historical socket statistical metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Column	Type	Description
`total_sockets_used`	int	Total sockets used in the system.
`tcp_sockets_inuse`	int	Number of TCP sockets in use.
`tcp_sockets_orphan`	int	Number of TCP sockets orphaned.
`tcp_sockets_timewait`	int	Number of TCP sockets in Time-Wait.
`tcp_sockets_alloc`	int	Number of TCP sockets allocated.
`tcp_sockets_memusage_inbytes`	int	Amount of memory consumed by TCP sockets.
`udp_sockets_inuse`	int	Number of UDP sockets in use.
`udp_sockets_memusage_inbytes`	int	Amount of memory consumed by UDP sockets.
`raw_sockets_inuse`	int	Number of RAW sockets in use.
`frag_sockets_inuse`	int	Number of FRAG sockets in use.
`frag_sockets_memusage_inbytes`	int	Amount of memory consumed by FRAG sockets.

system_*

The `system_*` tables store system utilization metrics. There are three system tables, all having the same columns:

- `system_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current system utilization data is stored in `system_now` during the period between data collection from the Command Center agents and automatic commitment to the `system_history` table.
- `system_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for system utilization data that has been cleared from `system_now` but has not yet been committed to `system_history`. It typically only contains a few minutes worth of data.
- `system_history` is a regular table that stores historical system utilization metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Column	Type	Description
`ctime`	timestamp	Time this row was created.
`hostname`	varchar(64)	Segment or master hostname associated with these system metrics.
`mem_total`	bigint	Total system memory in Bytes for this host.
`mem_used`	bigint	Used system memory in Bytes for this host.
`mem_actual_used`	bigint	Used actual memory in Bytes for this host (not including the memory reserved for cache and buffers).
`mem_actual_free`	bigint	Free actual memory in Bytes for this host (not including the memory reserved for cache and buffers).
`swap_total`	bigint	Total swap space in Bytes for this host.
`swap_used`	bigint	Used swap space in Bytes for this host.
`swap_page_in`	bigint	Number of swap pages in.
`swap_page_out`	bigint	Number of swap pages out.
`cpu_user`	float	CPU usage by the Greenplum system user.
`cpu_sys`	float	CPU usage for this host.
`cpu_idle`	float	Idle CPU capacity at metric collection time.
`load0`	float	CPU load average for the prior one-minute period.
`load1`	float	CPU load average for the prior five-minute period.
`load2`	float	CPU load average for the prior fifteen-minute period.
`quantum`	int	Interval between metric collection for this metric entry.
`disk_ro_rate`	bigint	Disk read operations per second.
`disk_wo_rate`	bigint	Disk write operations per second.
`disk_rb_rate`	bigint	Bytes per second for disk write operations.
`net_rp_rate`	bigint	Packets per second on the system network for read operations.
`net_wp_rate`	bigint	Packets per second on the system network for write operations.
`net_rb_rate`	bigint	Bytes per second on the system network for read operations.
`net_wb_rate`	bigint	Bytes per second on the system network for write operations.

tcp_stats_*

The `tcp_stats_*` tables store statistical metrics about TCP communications for a Greenplum Database instance.

These tables are in place for future use and are not currently populated.

There are three system tables, all having the same columns:

- `tcp_stats_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`.
- `tcp_stats_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for TCP statistical data that has been cleared from `tcp_stats_now` but has not yet been committed to `tcp_stats_history`. It typically only contains a few minutes worth of data.
- `tcp_stats_history` is a regular table that stores historical TCP statistical data. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Column	Type	Description
`segments_received`	bigint	Number of TCP segments received.
`segments_sent`	bigint	Number of TCP segments sent.
`segments_retransmitted`	bigint	Number of TCP segments retransmitted.
`active_connections`	int	Number of active TCP connections.
`passive_connections`	int	Number of passive TCP connections.
`failed_connection_attempts`	int	Number of failed TCP connection attempts.
`connections_established`	int	Number of TCP connections established.
`connection_resets_received`	int	Number of TCP connection resets received.
`connection_resets_sent`	int	Number of TCP connection resets sent.

udp_stats_*

The `udp_stats_*` tables store statistical metrics about UDP communications for a Greenplum Database instance.

These tables are in place for future use and are not currently populated.

There are three system tables, all having the same columns:

- `udp_stats_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`.
- `udp_stats_tail` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. This is a transitional table for UDP statistical data that has been cleared from `udp_stats_now` but has not yet been committed to `udp_stats_history`. It typically only contains a few minutes worth of data.
- `udp_stats_history` is a regular table that stores historical UDP statistical metrics. It is pre-partitioned into monthly partitions. Partitions are automatically added in two month increments as needed. Administrators must drop old partitions for the months that are no longer needed.

Column	Type	Description
`packets_received`	bigint	Number of UDP packets received.
`packets_sent`	bigint	Number of UDP packets sent.
`packets_received_unknown_port`	int	Number of UDP packets received on unknown ports.
`packet_receive_errors`	bigint	Number of errors encountered during UDP packet receive.

iterators_*_rollup

The `iterators_*_rollup` set of views aggregate the metrics stored in the `iterators_*` tables. A query iterator refers to a node or operation in a query plan. For example, a sequential scan operation may be one type of iterator in a particular query plan. For each iterator in a query plan, the `iterators_*` tables store the metrics collected from each segment instance. The `iterators_*_rollup` views summarize the query iterator metrics across all segments in the system.

The `tmid` , `ssid` and `ccnt` columns are the composite key that uniquely identifies a particular query.

There are three iterators rollup views, all having the same columns:

- The `iterators_now_rollup` view shows iterator data from the `interators_now` table aggregated across all segments in the system.
- The `iterators_tail_rollup` view shows iterator data from the `interators_tail` table aggregated across all segments in the system.
- The `iterators_history_rollup` shows iterator data from the `interators_history` table aggregated across all segments in the system.

See also the `iterators_*` tables for more information about the query plan iterator types and the metrics collected for each iterator.

`sample_time`	timestamp	The `ctime` from the associated `iterators_*` table.
`tmid`	int	A time identifier for a particular query. All iterator records associated with the query will have the same `tmid`.
`ssid`	int	The session id as shown by the `gp_session_id` parameter. All iterator records associated with the query will have the same `ssid`.
`ccnt`	int	The command number within this session as shown by `gp_command_count` parameter. All iterator records associated with the query will have the same `ccnt`.
`nid`	int	The ID of this query plan node from the slice plan.
`pnid`	int	The `pnid` (slice plan parent node ID) from the associated `iterators_*` table.
`ntype`	text	The `ntype` (node/iterator type) from the associated `iterators_*` table.
`nstatus`	text	The accumulated status of this iterator. Possible values are: Initialize, Executing, or Finished.
`tstart`	timestamp	The average start time for this iterator.
`tduration`	numeric	The average execution time for this iterator.
`pmemsize`	numeric	The average work memory allocated by the Greenplum planner to this iterator’s query processes.
`pmemmax`	numeric	The average of the maximum planner work memory used by this iterator’s query processes.
`memsize`	numeric	The average OS memory allocated to this iterator’s processes.
`memresid`	numeric	The average resident memory allocated to this iterator’s processes (as opposed to shared memory).
`memshare`	numeric	The average shared memory allocated to this iterator’s processes.
`cpu_elapsed`	numeric	Sum of the CPU usage of all segment processes executing this iterator.
		The current average percentage of CPU

`cpu_currpct`	double precision	utilization used by this iterator's processes. This value is always zero for historical (completed) iterators.
`rows_out`	numeric	The total number of actual rows output for this iterator on all segments.
`rows_out_est`	numeric	The total number of output rows for all segments as estimated by the query planner.
`skew_cpu`	numeric	Coefficient of variation for `cpu_elapsed` of iterators across all segments for this query, multiplied by 100. For example, a value of .95 is rendered as 95.
`skew_rows`	numeric	Coefficient of variation for `rows_out` of iterators across all segments for this query, multiplied by 100. For example, a value of .95 is rendered as 95.
`m0`	text	The name (`m0_name`), unit of measure (`m0_unit)` , average actual value (`m0_val`), and average estimated value (`m0_est`) for this iterator metric across all segments. The `m0` metric is always rows for all iterator types.
`m1`	text	The name (`m1_name`), unit of measure (`m1_unit`), average actual value (`m1_val`), and average estimated value (`m1_est`) for this iterator metric across all segments.
`m2`	text	The name (`m2_name`), unit of measure (`m2_unit`), average actual value (`m2_val`), and average estimated value (`m2_est`) for this iterator metric across all segments.
`m3`	text	The name (`m3_name`), unit of measure (`m3_unit`), average actual value (`m3_val`), and average estimated value (`m3_est`) for this iterator metric across all segments.
`m4`	text	The name (`m4_name`), unit of measure (`m4_unit`), average actual value (`m4_val`), and average estimated value (`m4_est`) for this iterator metric across all segments.
`m5`	text	The name (`m5_name`), unit of measure (`m5_unit`), average actual value (`m5_val`), and average estimated value (`m5_est`) for this iterator metric across all segments.
`m6`	text	The name (`m6_name`), unit of measure (`m6_unit`), average actual value (`m6_val`), and average estimated value (`m6_est`) for this iterator metric across all segments.
`m7`	text	The name (`m7_name`), unit of measure (`m7_unit`), average actual value (`m7_val`), and average estimated value (`m7_est`) for this iterator metric across all segments.
`m8`	text	The name (`m8_name`), unit of measure (`m8_unit`), average actual value (`m8_val`), and average estimated value (`m8_est`) for this iterator metric across all segments.
`m9`	text	The name (`m9_name`), unit of measure (`m9_unit`), average actual value (`m9_val`), and average estimated value (`m9_est`) for this iterator metric across all segments.
`m10 - m5`	text	Metrics `m10` through `m15` are not currently used by any iterator types.

`t0`	text	The name of the relation (`t0_val`) being scanned by this iterator. This metric is collected only for iterators that perform scan operations such as a sequential scan or function scan.

dynamic_memory_info

The `dynamic_memory_info` view shows a sum of the used and available dynamic memory for all segment instances on a segment host. Dynamic memory refers to the maximum amount of memory that Greenplum Database instance will allow the query processes of a single segment instance to consume before it starts cancelling processes. This limit is set by the `gp_vmem_protect_limit` server configuration parameter, and is evaluated on a per-segment basis.

Column	Type	Description
`ctime`	timestamp(0) without time zone	Time this row was created in the `segment_history` table.
`hostname`	varchar(64)	Segment or master hostname associated with these system memory metrics.
`dynamic_memory_used_mb`	numeric	The amount of dynamic memory in MB allocated to query processes running on this segment.
`dynamic_memory_available_mb`	numeric	The amount of additional dynamic memory (in MB) available to the query processes running on this segment host. Note that this value is a sum of the available memory for all segments on a host. Even though this value reports available memory, it is possible that one or more segments on the host have exceeded their memory limit as set by the `gp_vmem_protect_limit` parameter.

memory_info

The memory_info view shows per-host memory information from the system_history and segment_history tables. This allows administrators to compare the total memory available on a segment host, total memory used on a segment host, and dynamic memory used by query processes.

Column	Type	Description
`ctime`	timestamp(0) without time zone	Time this row was created in the `segment_history` table.
`hostname`	varchar(64)	Segment or master hostname associated with these system memory metrics.
`mem_total_mb`	numeric	Total system memory in MB for this segment host.
`mem_used_mb`	numeric	Total system memory used in MB for this segment host.
`mem_actual_used_mb`	numeric	Actual system memory used in MB for this segment host.
`mem_actual_free_mb`	numeric	Actual system memory free in MB for this segment host.
`swap_total_mb`	numeric	Total swap space in MB for this segment host.
`swap_used_mb`	numeric	Total swap space used in MB for this segment host.
`dynamic_memory_used_mb`	numeric	The amount of dynamic memory in MB allocated to query processes running on this segment.
`dynamic_memory_available_mb`	numeric	The amount of additional dynamic memory (in MB) available to the query processes running on this segment host. Note that this value is a sum of the available memory for all segments on a host. Even though this value reports available memory, it is possible that one or more segments on the host have exceeded their memory limit as set by the `gp_vmem_protect_limit` parameter.