PRODUCT DOCUMENTATION

Pivotal™ Greenplum Command Center

Version 2.2

Administrator Guide

Rev: A01

Notice

Copyright

Revised May 2016 (2.2.0)

Contents

Chapter 1: Overview....................5

Chapter 2: Setting Up Greenplum Command Center....................8
- Data Computing Appliance (DCA) Environments....................9
- Greenplum Database Software Environments.................... 10
- Enabling the Data Collection Agents....................11
- Installing the Greenplum Command Center Console....................13
  - Install the Command Center Console Software.................... 13
  - Set Up the Command Center Console Instance....................14
- About the Command Center Installation.................... 19
- Connecting to the Greenplum Command Center Console....................20
- Configuring Authentication for the Command Center Console.................... 21
  - Setting Up Secure Two-Way Authentication for Greenplum Command Center.................... 22
- Enabling Multi-Cluster Support....................30
- Upgrading Greenplum Command Center....................32
- Uninstalling Greenplum Command Center....................34

Chapter 3: Administering Greenplum Command Center....................35
- Starting and Stopping Greenplum Command Center.................... 36
  - Starting and Stopping Command Center Agents....................36
  - Starting and Stopping Command Center Console....................36
- Configuring Greenplum Command Center....................37
  - Agent Configuration....................37
  - Console Configuration....................37
  - Changing the gpmon Password.................... 37
- Administering Command Center Agents.................... 38
  - Adding and Removing Hosts....................38
  - Viewing and Maintaining Master Agent Log Files.................... 38
- Administering the Command Center Database....................39
  - Connecting to the Command Center Database.................... 39
  - Backing Up and Restoring the Command Center Database....................39
  - Maintaining the Historical Data Tables....................39
- Web Server Administration....................40
  - Configuring the Web Server.................... 40
  - Viewing and Maintaining Web Server Log Files....................40

Chapter 4: Utility Reference.................... 41
- gpperfmon_install.................... 42
- gpcmdr....................44

Chapter 5: Configuration File Reference....................46
- Command Center Agent Parameters....................47
- Command Center Console Parameters.................... 49
  - Web Application Parameters.................... 49
  - Web Server Parameters.................... 49

Greenplum Database Server Configuration Parameters....51
Setup Configuration File....52

Chapter 6: Command Center Database Reference.... 54
database_*.... 56
emcconnect_history.... 57
diskspace_*.... 59
filerep_*.... 60
health_*.... 66
interface_stats_*....67
iterators_*....69
Iterator Metrics.... 74
log_alert_*.... 91
queries_*....93
segment_*.... 96
socket_stats_*....97
system_*.... 98
tcp_stats_*.... 100
udp_stats_*.... 101
iterators_*_rollup....102
dynamic_memory_info....106
memory_info....107

Chapter 1

Overview

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 management tasks such as start, stop, and recovery of systems in a Greenplum environment. The Greenplum Command Center Console is an interactive graphical web application that may be installed on a web server 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 send it to the Greenplum master host at regular intervals. Greenplum Command Center stores its data and metrics in a dedicated Greenplum database (the Command Center database) whose information is distributed among the master host and segment hosts, like any other Greenplum Database. You can access the data stored in the Greenplum 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 Data Integration Accelerator (DIA) Module
- Greenplum Data Computing Appliance Hardware

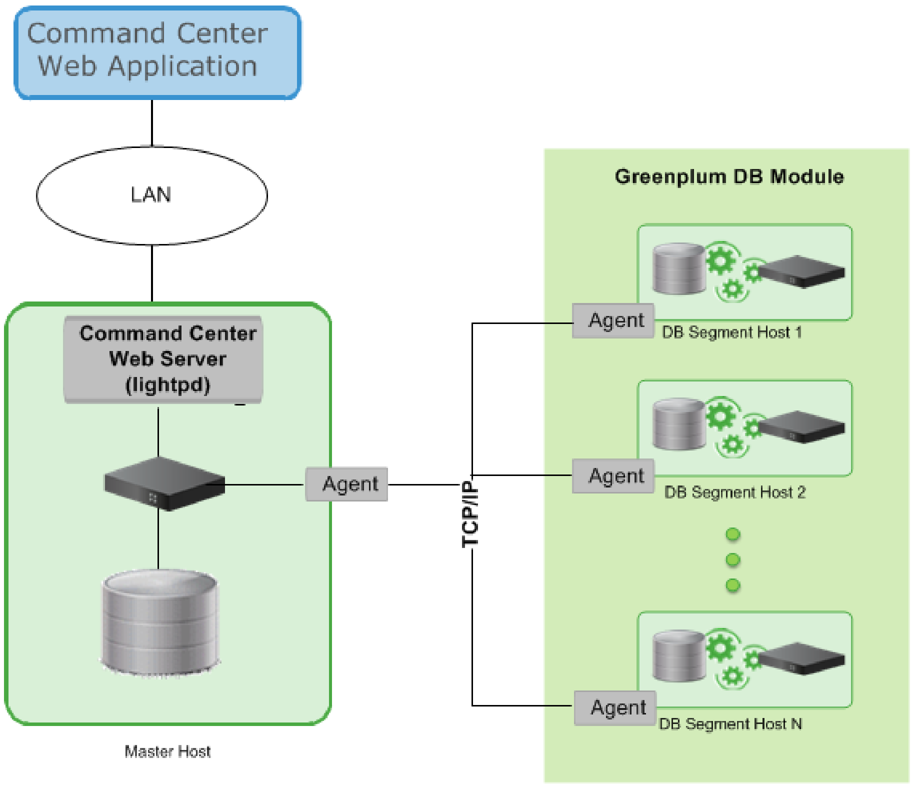
Greenplum Database (software-only environments):

- 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

You can enable or disable Command Center using the `gp_enable_gpperfmon` server configuration parameter. After it is enabled, 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 installation includes setup scripts to install the Command Center database (gpperfmon).

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 managment 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 graphical console for viewing Greenplum System metrics and performing certain database administrative tasks. This browser-based application provides the following functionality:

Database administrative controls

- Ability to stop/start the database
- Ability to recover/rebalance segments

An interactive view of system metrics

- Realtime
- Historic (configurable by time)

An interactive view of system hardware health

- This functionality is only available for Greenplum Data Computing Appliance environments.

Database query monitoring

- Ability to view, search, prioritize, or cancel any query in the system.
- Ability to view the internals of any query, including the query plan, query plan iterator-level details, and real-time information about individual scans and joins.

Database query monitoring

- Ability to view, search, prioritize, or cancel any query in the system.
- Ability to view the internals of any query, including the query plan, query plan iterator-level details, and real-time information about individual scans and joins.

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 *Installing the Greenplum Command Center Console*.

Greenplum Command Center Web Service

The Greenplum Command Center Console queries the Command Center database through a web service framework composed of a lightweight lighttpd web server and Python-based middleware. The lighttpd server is an open-source web server with a low memory footprint and light CPU load. For more information, see *http://www.lighttpd.net/*.

The console setup utility sets up the lighttpd 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*.

Chapter 2

Setting Up Greenplum Command Center

Information about installing and setting up Pivotal Greenplum Command Center and Pivotal Greenplum Workload Manager.

Note: The Greenplum Workload Manager installer is included in the Greenplum Command Center installer. You can install Workload Manager as an option when you create a Greenplum Command Center instance. Alternatively, you can install Workload Manager separately from the command line using the bundled installer. See the *Greenplum Workload Manager User Guide* for instructions to run the Greenplum Workload Manager installer.

- *Data Computing Appliance (DCA) Environments*
- *Greenplum Database Software Environments*
- *Enabling the Data Collection Agents*
- *Installing the Greenplum Command Center Console*
- *About the Command Center Installation*
- *Connecting to the Greenplum Command Center Console*
- *Configuring Authentication for the Command Center Console*
- *Enabling Multi-Cluster Support*
- *Upgrading Greenplum Command Center*
- *Uninstalling Greenplum Command Center*

Data Computing Appliance (DCA) Environments

The Greenplum Data Computing Appliance (DCA) version of Pivotal Greenplum Command Center is already installed on the appliance (versions 1.2.x and 2.x).

For more information about setting up and configuring Greenplum Command Center on a Greenplum DCA, refer to the relevant versions of:

- *Greenplum Data Computing Appliance Software Upgrade Guide*
- *Greenplum Data Computing Appliance Installation and Configuration Guide.*

Greenplum Database Software Environments

If you are using Greenplum Command Center in a software-only environment, installing and enabling Greenplum Command Center is a two-step process. First, you create the Greenplum Command Center database (gpperfmon) and enable the data collection agents within your Greenplum system. After data collection is enabled, the next (optional) step is to install and configure the Greenplum Command Center Console (the Web application used to view the Command Center data stored in Greenplum Database).

Greenplum Command Center software for Greenplum Database software-only environments is available as a separate download from *Pivotal Network*. Installer files are available for Red Hat Enterprise Linux 64-bit.

Enabling the Data Collection Agents

This section describes how to create the Command Center database and enable the Command Center data collection agents. When the data collection agents are enabled, their processes are started and stopped along with the Greenplum Database server processes (using `gpstart` and `gpstop`).

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 Command Center superuser role (edits the `pg_hba.conf` and `.pgpass` files).
- Sets the Command Center server configuration parameters in the Greenplum Database server `postgresql.conf` files.

To enable the Collection Agents

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

```
$ su - gpadmin
```

2. Source the path file from your master host's 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 `gperfmon_install` utility adds the `gpmon` user to the `pg_hba.conf` authorization configuration file with an entry that permits local connections to any database. Since the `gpmon` role is a Greenplum Database superuser, you may wish to restrict the role to just the gpperfmon database after you run `gpperfmon_install`. Edit the `$MASTER_DATA_DIRECTORY/pg_hba.conf` file and change this line:

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

to the following:

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

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
```

To configure 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`).

Installing the Greenplum Command Center Console

The Command Center Console provides a graphical interface for viewing performance data and for administering certain aspects of your Greenplum system.

The Command Center Console is typically installed on the Greenplum Database master host. However, you have the option to install the console on a host different from the master host. Note that this setup incurs a performance penalty due to the numerous database connections the console must open over the network.

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.

The Command Center Console supports browsers that have at a minimum Adobe Flash 9.0 (for GPCC 1.2.2.2 and earlier) or Adobe Flash 11.0 (for GPCC 1.2.2.3 and later) enabled.

> Important:
>
> We recommend that you always install the latest version of Adobe Flash Player to ensure you receive the latest security updates from Adobe.

For example, the following browsers are supported:

- Internet Explorer for Windows XP and Vista
- Mozilla Firefox for Windows and Linux
- Apple Safari browser for Macintosh
- Google Chrome

The Command Center Console runs on a lighttpd web server. The default web server port is 28080. For more information about the web server, see *Web Server Administration*.

Installing the Command Center Console involves the following high-level tasks:

- *Install the Command Center Console Software* – Create the software installation directory.
- *Set Up the Command Center Console Instance* – Set up the environment variables and then create and configure a Command Center Console instance and its supporting web services.

Install the Command Center Console Software

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 also install Greenplum Database installation binary files on the remote system. After installing the binary files, source `greenplum_path.sh`, then perform the Console installation steps described below. Note that you do not need to initialize the database. See the *Greenplum Database Installation Guide* for more information.

1. Download the installer file from *Pivotal Network*. Installer files are available for the Red Hat Enterprise Linux 64-bit platform. You do not need to download the installer file if you are installing the console on an EMC Data Computing Appliance; the installer file is already loaded on DCAs.
2. Unzip the installer file where *`PLATFORM`* is `RHEL5-x86_64` (Red Hat 64-bit). For example:

   ```
   # unzip greenplum-cc-web-1.3.0.0-RHEL5-x86_64.zip
   ```
3. Log in as `gpadmin`.
4. Launch the installer using `bash`. For example:

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

5. Read through the license agreement. When you reach the bottom, type `yes` to accept the license agreement.
6. The installer prompts you to provide an installation path. Press Enter to accept the default install path (`/usr/local/greenplum-cc-web-x.x.x.x`), or enter an absolute path to an install location. You must have write permissions to the location you specify.
7. Once the installation has completed successfully, create a host file listing all remaining hostnames, including the standby master host. Hostnames must be DNS resolvable.
8. The installation directory contains a `gpcc_path.sh` file with path and environment settings for the Console. Source this and the Greenplum path, as follows:

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

   Note:

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

   Change the ownership of the installation directory:

   ```
   $ chown -R gpadmin:gpadmin greenplum-cc-web-x.x.x.x
   ```

   Change the permissions of the installation directory:

   ```
   $ chmod -R 755 greenplum-cc-web-x.x.x.x
   ```
9. 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 earlier in this procedure.
10. Configure the Console as described in *Set Up the Command Center Console Instance*.

Set up the Command Center Environment

Follow the steps below to set up the Greenplum Command Center environment for the `gpadmin` user.

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-x.x.x.x
   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
   ```

Set Up the Command Center Console Instance

A Command Center instance is a connection to a Greenplum Database cluster. The `gpcmdr --setup` command sets up the Command Center Console. The command can be run interactively, or you can create an installation configuration file to run the installation non-interactively. When you use a configuration file, you can create multiple Command Center instances at once.

Command Center instances are typically set up on the Greenplum master host; if installed on another host, the console experiences slower performance due to frequent connections to the gpperfmon database.

If you choose to secure web browser connections to the Command Center web server with SSL, you can provide a server certificate or allow the `gpcmdr` command to generate a self-signed certificate for you. Because the generated certificate is self-signed, clients cannot verify that the certificate is signed by a trusted Certificate Authority, so they must override an exception on their first connection to the web server. This can be avoided if you supply a certificate signed by a commercial or local Certificate authority. The SSL configuration also enables Diffie-Hellman key exchange, which requires a dhparams file. This file can be generated by the `gpcmdr` or supplied by you. See *Acquire or Create an SSL Certificate (Optional)* for instructions.

Configuration files, log files, and runtime files for each Command Center instance are managed in a subdirectory of the `$GPPERFMON/instances` directory.

- *Acquire or Create an SSL Certificate (Optional)*
- *Set up a Greenplum Command Center Instance*
- *Setting Up Command Center Instances with a Configuration File*

Acquire or Create an SSL Certificate (Optional)

It is recommended that you enable SSL for the lighttpd Web server that serves the Command Center Console. SSL ensures that client connections to the Greenplum Command Center are negotiated securely and encrypted. To enable SSL, you will need a server certificate for the web server and a Diffie-Hellman parameters (dhparam) file, which is used while negotiating the connection.

You can use an existing certificate and dhparam file or you can create a self-signed certificate and a dhparam file when you set up a Command Center Console instance. If you use a self-signed certificate, Command Center users will have to explicitely override an exception when they first browse to the Control Center URL, since the certificate is not signed by a trusted CA. However, the connection is still effectively encrypted.

Ideally, you should acquire a signed certificate from a commercial Certificate Authority or your organization's internal Certificate Authority. If you already have a certificate and dhparam file, install them on the server where GPCC is installed, for example in the `/etc/ssl/certs` directory. Then you can choose to import them when you create a Control Center instance.

If you want to enable SSL in an existing Control Center instance, you can create the certificate and dhparam files yourself and add the SSL parameters to the `instances/`*`instance-name`*`/conf/lighttpd.conf` file.

This is the recommended SSL configuration for the lighttpd web server:

```
ssl.engine = "enable"
ssl.pemfile = "/path/to/cert.pem"
ssl.dh-file = "/path/to/dhparam.pem"
ssl.ec-curve = "secp384r1"
ssl.use-sslv2 = "disable"
ssl.use-sslv3 = "disable"
ssl.honor-cipher-order = "enable"
ssl.use-compression = "disable"
ssl.cipher-list = "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH"
```

The following example creates certificate and dhparam files in the `/etc/ssl/certs` directory. Note that because the certificate is self-signed, users will have to override the SSL exception to proceed to the Control Center. Perform these steps as `root`.

1. Create a certificate for the Web server.

   ```
   # cd /etc/ssl/certs
   # openssl req -newkey rsa:2048 -x509 -keyout cert.pem -out cert.pem -days 3650 -
   nodes
   ```

Enter the requested distinguished name (DN) information at the prompts to create an unsigned certificate. For example:

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) [Berkshire]:California
Locality Name (eg, city) [Newbury]:Palo Alto
Organization Name (eg, company) [My Company Ltd]:Pivotal Software, Inc.
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:mdw
Email Address []:
```

2. Create a Diffie-Hellman parameters file. This command can take a long time to finish.

```
# cd /etc/ssl/certs
# openssl dhparam -out dhparam.pem 4096
```

Set up a Greenplum Command Center Instance

Follow the steps below to run the `gpcmdr --setup` command to create an instance. To accept the displayed default values for any parameters at configuration time, press the ENTER key. To monitor multiple instances, run the setup utility separately for each instance.

1. Log in as the Greenplum administrator (`gpadmin`).
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. 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 of the master at the prompt.
5. 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.
6. Provide the port for the Greenplum Database master instance.
7. Enter `y` to install Greenplum Workload Manager, or `n` if you do not want to install Workload Manager now. If you enter `y`, Workload Manager is installed into the current user's home directory, `$HOME/gp-wlm`.

   Installing Workload Manager with `gpcmdr` is only supported when `gpcmdr` is running on the master host. If you answered `y` in step 4 to specify that the Greenplum Database master is on a remote host, you must enter `n` for this step. You can install Workload Manager on the master host later, after creating the Command Center instance.

   Note:

   You can run the Workload Manager installer separately at the command line. Command-line installation allows you to override the Workload Manager installer defaults, such as the installation location. See "Installing Greenplum Workload Manager" in the *Pivotal Greenplum Workload Manager User Guide* for instructions.

   In rare instances, the Workload Manager installer will fail during the `cluster-health-check` phase. If the installer reports that the cluster is not healthy, run `gpcmdr --setup` without installing Workload Manager, and then run the Workload Manager installer at the command line with the `--force` option. When you re-run `gpcmdr --setup` you must either provide a new instance name or remove the instance directory from the `$GPPERFMONHOME/instances` directory.
8. Provide a port number for the Command Center Console web server. The default is 28080.
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, we recommend you use SSL to encrypt these communications.

If you choose to enable SSL:

a. You are asked if you want to import a certificate file. If you have a certificate you want to use, enter `y`, then enter the full path to the certificate file. The path you enter is added to the `lighttp.conf` file.
b. You are asked if you want to import the dhparams file. If you have a dhparams file to use, enter `y`, then enter the full path to the file. The path you enter is added to the `lighttp.conf` file.
c. If you did not import a certificate, `gpcmdr` generates a private key and then prompts you to enter the Distinguished Name information needed to generate a Certificate Signing Request (CSR). For example:

```
Country Name (2 letter code) [GB]:US
State or Province Name (full name) []:California
Locality Name (eg, city) [Default City]:Palo Alto
Organization Name (eg, company) [My Company Ltd]:Pivotal Software, Inc.
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:mdw
Email Address []:
```

The CSR is signed with the previously generated private key and the certificate is saved in the `instances/`*`instance-name`*`/conf` directory.
d. If you did not import a dhparams file, `gpcmdr` generates one and saves it in the `instances//conf` directory. This may take a long time.

10. Enter `y` to enable IPv6 support. IPv6 support is disabled by default.
11. Enter `y` to enable cross site request forgery protection for the GPCC Web API. This is disabled by default.
12. 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.
13. Update the `pg_hba.conf` file to allow the `gpmon` role access to every database that will be monitored using Control Center. Specify the md5 authentication method. This example allows `gpmon` access to all databases when GPCC is running on the master host:

```
local       all     gpmon      md5
```

14. Start the Console and log in. See *Connecting to the Greenplum Command Center Console*.
15. 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.

Setting Up Command Center Instances with a Configuration File

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 a Python ConfigParser file, which is similar to a Windows INI file. The configuration file contains one or more sections, each section 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_ipv6: false
enable_csrf_protect: true
enable_copy_standby: true
standby_master_host: smdw
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
install_wlm: false

[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
```

If you want to install just one instance, you can add the section header to the command. For example, the following command installs the `dev` instance:

```
gpcmdr --setup development --config_file myconfig.cfg
```

If you enable SSL and do not provide an SSL certificate, `gpcmdr` will run the `openssl` command to create a certificate, which requires input from the user. To avoid the need for user input, be sure to set the `enable_user_import_cert` and `ssl_cert_file` parameters.

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

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
- `ext` – Python directory and files
- `instances` – contains a subdirectory of resources for each Greenplum Database instance monitored by the console
- `lib` – library files for Greenplum Command Center
- `www` – web service 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:

- `lighttpd.pid` – file containing an updated process ID for the web server process for the instance
- `conf` – console and web server configuration files, `gpperfmonui.conf` and `lighttpd.conf`
- `logs` – logs for the web server for this instance
- `perfmon.fastcgi.socket-0` – dynamically updated socket information, which cannot be viewed or updated
- `sessions` – files storing information about session state
- `tmp` – temporary files used by the web server
- `web` – symbolic links to web server files in the installation directory

Connecting to the Greenplum Command Center Console

Start the Greenplum Command Center Console 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.

After the instance is running, you can 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, use the following web address:

```
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 Command Center Console online help for more information.

You must be a Greenplum administrator to fully use the Greenplum Command Center Console. Administrators can view information for all queries, as well as system metrics, while regular database users can only monitor their own queries.

Configuring Authentication for the Command Center Console

The installation utility created the Greenplum Command Center database, enabled the data collection agents, and created a `gpmon` superuser. This is the Greenplum role used to manage the Command Center components and data within the Greenplum environment. The `gpmon` role is configured to use md5-encrypted password authentication to connect to the Greenplum Database instance. The `gpmon` role must be configured in `pg_hba.conf` to allow access to every database that will be monitored using the Command Center.

Greenplum Command Center does not accept logins from the `gpadmin` user, or from local users configured with `trust` authentication in the `pg_hba.conf` file. Allowing `trust` authentication for remote logins is discouraged because it is insecure.

There are three user levels in Greenplum Command Center.

- Regular user – Regular users may only view their own database queries and do not have access to the Administrative tab.
- Operator – Operators have access to more functionality in the Command Center Console than regular users, but they do not have to be a Greenplum Database superuser. Operators can view and cancel all queries and have limited access to administrative tasks. The Operator role must be created and users must be assigned to that role. The procedures are described below.
- Superuser – A Greenplum Database superuser can use all GPCC features, including viewing information for all database queries, system metrics, and administrative tasks.

The Command Center Console is configured by default to require md5-encrypted password authentication, so make sure each GPCC user role has an md5-encrypted password set.

If you are using Greenplum Database version 4.2.1 or higher, you have the option of using SHA-256-encrypted password authentication. You can specify SHA-256 authentication by changing the `password_hash_algorithm` server parameter. This parameter can be set either system-wide or on a session level.

Any other Greenplum Database users with appropriate privileges can access Command Center.

To create a new Command Center user, first you have to create a Greenplum Database user, then edit the `pg_hba.conf` file to give that user access to Command Center.

The following are steps to create new Command Center users.

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:
   To create a regular user - a database read-only role:

   ```
   # CREATE ROLE cc_user WITH LOGIN PASSWORD 'new_password';
   ```

   To create an Operator user - a database role that is a member of the `gpcc_operator` role:

   a. Create the role `gpcc_operator`:

      ```
      # CREATE ROLE gpcc_operator;
      ```

b. Grant Operator permissions to a user by making the user a member of the `gpcc_operator` role:

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

c. Grant Operator permissions to a group by granting the `gpcc_operator` role to the group:

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

To create a superuser - a database role with superuser privileges:

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

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

```
# \du
```

The new users you just created should be returned along with the attributes you specified.

5. Exit psql.

```
# \q
```

6. Edit the `pg_hba.conf` file to give the new user access to Command Center. Open the file in an editor:

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

7. 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
```

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.

8. Save the file and exit the editor.
9. Enter the following command to reload Greenplum Database processes.

```
# gpstop -u
```

Setting Up Secure Two-Way Authentication for Greenplum Command Center

This topic helps you to set up Greenplum Command Center to authenticate users with digital certificates and to encrypt connections between users' web browsers and the Command Center web server.

Greenplum Command Center is a browser-based application that, by default, uses basic authentication to allow connections. When the user browses to the Greenplum Command Center web server, the server prompts for a user name and password. The user enters their Greenplum Database role name and password and the server uses it to log in to the *gpperfmon* database. The password is md5-encoded, but the data transferred over the network is otherwise unencrypted.

To encrypt data over the connection requires a secure, two-way authenticated connection between the user's browser and the Greenplum Command Center web server. The Command Center web server and users' browsers are configured with X.509 certificates. The client uses the server's certificate to verify they have connected to the correct server and the server uses the client's certificate to verify the user's identity and to look up the user's Greenplum Database role name. Once the certificates are installed in the server and users' browsers, connections are established automatically when users browse to the Command Center URL.

Certificate-based authentication requires that both the server and client certificates are digitally signed by a trusted Certificate Authority (CA). Any web server accessible on the Internet should have a certificate signed by a well-known commercial CA, such as Symantec, to prevent man-in-the-middle attacks and other malicious attacks.

If your users and servers are confined to an intranet, you can set up a public key infrastructure (PKI) that allows you to act as the CA for your organization, or even for just the Greenplum Command Center. You create a public/private key pair for your CA, use it to generate a certificate signing request (CSR), and then sign it with your own certificate, resulting in a *self-signed certificate*. This certificate can then be used to sign CSRs for the Command Center web server and for GPCC users. The CA public key must be installed into users' web browsers, along with their own signed user certificates.

A user's certificate contains an attribute, Common Name (CN), which Command Center uses to look up the user's Greenplum Database role. The CN attribute is mapped to the database role in the *user_cert_mapping* table in the *gpperfmon* database. When the user connects to the Command Center web server, the Command Center web application logs in to the *gpperfmon* database with the `gpmon` role and looks up the CN in the *user_cert_mapping* table to find the user's Greenplum Database role.

> Note: When SSL is enabled for a Greenplum Command Center installation, all web browsers connecting to Greenplum Command Center must be configured with a client SSL certificate. In a multi-cluster configuration, all hosts must have the same SSL configuration. SSL must be enabled or disabled for all hosts. For information about multi-cluster configuration, see *Enabling Multi-Cluster Support*.

These tasks enable certificate-based authentication and encryption of Command Center sessions:

- *Setting Up an OpenSSL PKI*
- *Configuring the Command Center Web Server (lighttpd)*
- *Creating a Client SSL Certificate*
- *Configuring the GPCC Database (gpperfmon)*
- *Configuring a Web Browser*

This process sets up a simple PKI using OpenSSL. The OpenSSL `openssl` command-line utility is included with GPCC and is added to your path when you source the `gpcc_path.env` file. See the OpenSSL documentation for information about OpenSSL and the commands that are used.

Setting Up an OpenSSL PKI

The instructions in this section set up an OpenSSL public key infrastructure (PKI) that enables you to generate and sign Greenplum Command Center user certificates.

1. Log in to the server where you installed Greenplum Command Center and source the Greenplum environment files.

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

2. Change to the `$GPPERFMONHOME/etc` directory, where the `openssl.cnf` is located.

   ```
   $ cd $GPPERFMONHOME/etc
   ```

3. *(Optional)* The `openssl.cnf` file contains default settings for a Certificate Authority named `demoCA`. You can edit the file, or a copy of the file, and change the settings to suit your own organization. Refer to the *OpenSSL documentation* for help. Note that the `[Default_CA]` section defines directory and file names that are used in the following steps. If you change them, be sure to use the new values in the following commands.

4. Create a directory named `demoCA` at the location where the `openssl.cnf` file is located.

```
$ mkdir demoCA
```

5. Create the PKI directory structure in the `demoCA` directory.

```
$ mkdir demoCA/certs
$ mkdir demoCA/newcerts
$ mkdir demoCA/private
```

6. Create `serial`, `crlnumber` and `index.txt` files in the `demoCA` directory.

```
$ echo "00" >> demoCA/serial
$ echo "00" >> demoCA/crlnumber
$ touch demoCA/index.txt
```

7. Generate a private RSA key for the CA. This command generates an RSA 2048 bit private key for the CA:

```
$ openssl genrsa -out demoCA/cakey.key 2048
```

8. Use the generated private key to sign itself. You are prompted to enter the information about the CA you are establishing:

   - Country Name:The two-letter code for the country, with no punctuation.
   - State or Province: Spell out the state name completely.
   - Locality or City: The city or town name.
   - Company: The name of the company. If the company or department has an &, @, or any other symbol using the shift key in its name, spell out the symbol or omit it.
   - Organizational Unit: *(Optional)* Can be used to help identify certificates registered to an organization.
   - Common Name: The name of this CA.
   - Email Address: *(Optional)* The email address of the owner of this certificate.

   The following example creates a new self-signed X.509 certificate that is valid for ten years for the keypair in the `certAut.key` file. This key will be used to sign certificates generated for the web server and client Command Center users.

```
$ openssl req -new -x509 -days 3650 -key demoCA/cakey.key -out demoCA/cacert.crt -
config openssl.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Palo Alto
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Pivotal Software, Inc.
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:GPCC CA
Email Address []:
```

9. Move the CA key and certificate files into place in the `demoCA` directory. The names and locations of the files are specified in the `openssl.cnf` file.

```
$ mv demoCA/cakey.key demoCA/private/cakey.pem
$ mv demoCA/cacert.crt demoCA/cacert.pem
```

Configuring the Command Center Web Server (lighttpd)

1. Generate a private key for the Command Center web server.

   ```
   $ openssl genrsa -out demoCA/private/wskey.key 2048
   ```

2. Create a CSR for the web server's private key. You are prompted to enter information about the server where the certificate will be installed. Set the Common Name to the name of the host were the Command Center web server is running.

   - Country Name:The two-letter code for the country, with no punctuation.
   - State or Province: Spell out the state name completely.
   - Locality or City: The city or town name.
   - Company: The name of the company. If the company or department has an &, @, or any other symbol using the shift key in its name, spell out the symbol or omit it.
   - Organizational Unit: *(Optional)* Can be used to help identify certificates registered to an organization.
   - Common Name: The exact name of the web server.
   - Email Address: *(Optional)* The email address of the owner of this certificate.

   ```
   openssl req -new -key demoCA/private/wskey.key -out wskey.csr -config openssl.cnf
   Enter pass phrase for demoCA/private/wskey.key:
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   Country Name (2 letter code) [AU]:US
   State or Province Name (full name) [Some-State]:California
   Locality Name (eg, city) []:Palo Alto
   Organization Name (eg, company) [Internet Widgits Pty Ltd]:Pivotal Software, Inc.
   Organizational Unit Name (eg, section) []:
   Common Name (e.g. server FQDN or YOUR name) []:mdw.pivotal.lan
   Email Address []:

   Please enter the following 'extra' attributes
   to be sent with your certificate request
   A challenge password []:many_secret
   An optional company name []:
   ```

3. Sign the web server's private key using your CA certificate:

   ```
   $ openssl ca -in wskey.csr -out demoCA/certs/wscert.crt -config openssl.cnf
   Using configuration from openssl.cnf
   Check that the request matches the signature
   Signature ok
   Certificate Details:
           Serial Number: 0 (0x0)
           Validity
               Not Before: Jun  5 23:16:30 2015 GMT
               Not After : Jun  4 23:16:30 2016 GMT
           Subject:
               countryName               = US
               stateOrProvinceName       = California
               organizationName          = Pivotal Software, Inc.
               commonName                = mdw.pivotal.lan
           X509v3 extensions:
               X509v3 Basic Constraints:
                   CA:FALSE
               Netscape Comment:
                   OpenSSL Generated Certificate
               X509v3 Subject Key Identifier:
                   09:20:A4:74:43:12:72:24:C3:F3:14:34:7E:A8:3A:BD:42:CA:3B:0E
   ```

```
             X509v3 Authority Key Identifier:
                 keyid:4D:4A:F1:FA:50:B8:EA:19:D4:1F:6F:18:34:E6:B8:CD:26:61:71:96

Certificate is to be certified until Jun  4 23:16:30 2016 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

4. Create a PEM file containing the web server's private key and certificate. The lighttpd web server requires this file.

   ```
   $ cat demoCA/private/wskey.key demoCA/certs/wscert.crt > wscert.pem
   ```

5. Edit the lighttpd configuration file `lighttpd.conf` for the instance configured for Greenplum Command Center. The configuration file is located in `$GPPERFMONHOME/instances/`*`instance`*`/conf`, where *instance* is the value you specified when you created the instance.

   Add the following parameters to enable SSL-based client authentication:

   ```

   ssl.engine = "enable"
   ssl.ca-file = "/usr/local/greenplum-cc-web/etc/demoCA/cacert.pem"
   ssl.pemfile = "/usr/local/greenplum-cc-web/etc/wscert.pem"
   ssl.verifyclient.activate = "enable"
   ssl.verifyclient.enforce = "enable"
   ssl.verifyclient.username = "SSL_CLIENT_S_DN_CN"
   ```

   Note: For the `ssl.ca-file` and `ssl.pemfile` parameters, you must specify the fully qualified paths to the `cacert.pem` and `wscert.pem` files, respectively.

Enter `gpcmdr --restart` *`instance_name`* to restart the lighttpd web server after updating the configuration file.

Creating a Client SSL Certificate

Follow these steps for each GPCC user to create a signed certificate to install in the user's web browsers.

1. Open a command line terminal and source the Greenplum environment files.

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

2. Change to the `$GPPERFMONHOME/etc` directory. -pp

   ```
   $ cd $GPPERFMONHOME/etc
   ```

3. Generate a client private key by executing the following command. Replace *client* in this and following commands with a string such as the user's login name or database role name.

   ```
   $ openssl genrsa -out demoCA/private/client.key 2048
   ```

4. Generate a certificate signing request for the user's private key. The value you enter for the Common Name field must be unique for each user; it is used to map the user to their Greenplum Database role.

   ```
   $ openssl req -new -key demoCA/private/client.key -out client.csr -config
    openssl.cnf
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   ```

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Palo Alto
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Pivotal Software, Inc.
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:gpcc_user_1
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:many_secret
An optional company name []:
```

5. Sign the client certificate with the CA certificate:

```
$ openssl ca -in client.csr -out demoCA/certs/client.crt -config openssl.cnf -
policy policy_anything
Using configuration from openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Jun  5 23:46:39 2015 GMT
            Not After : Jun  4 23:46:39 2016 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = California
            localityName              = Palo Alto
            organizationName          = Pivotal Software, Inc.
            commonName                = gpcc_user_1
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                C9:1D:18:66:FB:0F:7E:FA:40:FB:EA:DF:94:B5:FD:3E:FC:FC:46:41
            X509v3 Authority Key Identifier:
                keyid:4D:4A:F1:FA:50:B8:EA:19:D4:1F:6F:18:34:E6:B8:CD:26:61:71:96

Certificate is to be certified until Jun  4 23:46:39 2016 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Note: The value for the `-days` option must be less than or equal to the value specified for the CA certificate.

6. Create a `.pem` file containing the client's private key and certificate.

```
$ cat demoCA/private/client.key demoCA/certs/client.crt > client.pem
```

7. Convert the signed client certificate to PKCS#12 format. The PKCS #12 format is an archive file format for storing many cryptographic objects as a single file. It is commonly used to bundle a private key with its X.509 certificate.

```
$ openssl pkcs12 -export -in demoCA/certs/client.crt -inkey demoCA/
private/client.key -certfile demoCA/cacert.pem -out client.p12
Enter Export Password:
Verifying - Enter Export Password:
```

Important: The export password is required when the client certificate is imported into a client web browser. See *Configuring a Web Browser*.

8. Send the `*client*.pem` file and the `*client*.p12` file to the GPCC user. Securely communicate the export password for the `.p12` file to the user.

Configuring the GPCC Database (gpperfmon)

When the GPCC web server is configured to use SSL based client authentication, the GPCC web server queries the *user_cert_mapping* table in the *gpperfmon* database as part of the authentication process. The *user_cert_mapping* table maps the client certificate user ID (the common name in the client certificate) with the *gpperfmon* user role.

You must create the *user_cert_mapping* table and populate it with the proper user information.

1. Create the *user_cert_mapping* table in the *gpperfmon* database.

```
# psql gpperfmon
=# CREATE TABLE public.user_cert_mapping(mapping text primary key, username text);
```

2. For each Greenplum Command Center user who accesses the *gpperfmon* database, a row must exist in the *user_cert_mapping* table. The *mapping* column contains the common name that you specified when creating the client certificate. The corresponding *username* column contains the *gpperfmon* user role.

   The format of the common name in the *mapping* column is `'*common_name*, *common_name*'`. The common name listed twice, separated by a comma and space. For this example, the common name is `gpcc_user1`, and the *gpperfmon* user role is `perfmon_user1`:

```
# INSERT INTO public.user_cert_mapping VALUES ('gpcc_user1, gpcc_user1',
 'perfmon_user1');
```

Configuring a Web Browser

When Greenplum Command Center is configured to use SSL, web browsers connecting to Greenplum Command Center must have a client certificate and CA certificate. You import the created client certificate and CA certificate to your browser so that it can forward the certificates to Greenplum Command Center when connecting to a Greenplum database from Greenplum Command Center.

Follow these steps to import the certificates into Firefox and Chrome browsers, or into the Mac OS X Keychain Access utility.

Mozilla Firefox

1. From the menu, select Preferences.
2. Click Advanced and then click the Certificates tab.
3. Click View Certificates.
4. In the Certificate Manager dialog, open the Authorities tab, and click Import. Select your CA certificate (for example, `cacrt.crt`).
   - In the Downloading Certificate dialog, check Trust this CA to identify websites.
5. In the Your Certificates tab, click Import and select your client certificate (for example `*client*.p12`).

   The import process prompts for a password for the client certificate. Enter the export password that you entered when you converted the client certificate to PKCS#12. See *Creating a Client SSL Certificate*.
6. Click OK to apply the configuration changes.

Google Chrome

1. From the customization menu, select Settings and then select Show Advanced Settings.
2. In the section HTTPS/SSL, click Manage certificates....
3. In the Manage Certificates dialog, open the Trusted Root Certification Authorities tab . Click Import and select your CA certificate (for example, `certAut.crt`).

4. Open the Personal tab. Click Import and select the client certificate (for example, `client.p12`).

   The import process prompts for a password the client certificate. Enter the export password that you entered when you converted the client certificate to PKCS#12. See *Creating a Client SSL Certificate*.
5. Click Ok to apply configuration changes.
6. Browse to the Greenplum Command Center URL, for example `https://mdw:28080`. A User Identification Request window will appear. Choose the client certificate you imported and click OK.

Mac OS X

Mac OS X has a built-in certificate manager, Keychain Access. Google Chrome, Safari, and Mozilla Firefox use Keychain Access for certificate management. You will be asked to authenticate as an admin user on your Mac several times while following these steps.

To begin, the user needs the `cacert.pem` CA certificate and the `.pem` file containing the user's private key and certificate.

1. Open Keychain Access (Applications > Utilities > Keychain Access.
2. Choose File > Import , select the CA `.pem` file for the CA you setup (`cacert.pem`), and click Open.
3. On the dialog window that displays, click Always Trust.
4. Choose File > Import , select the `.pem` file for the GPCC user you setup (*`client`*`.pem`), and click Open.
5. In a web browser, browse to the GPCC Command Center (`https://`*`gpcc_host`*`:28080`).
6. Choose the GPCC client certificate in the Select a certificate dialog and click OK.

Enabling Multi-Cluster Support

Multi-cluster functionality allows you to view the status of multiple clusters, optionally categorized into groups, at one time in the GPCC UI.

Greenplum Command Center provides multi-cluster support beginning in release 1.3.0.

Multi-cluster support in the UI is configured by a superuser, using a configuration file.

From the GPCC UI, click the Multi-Cluster link in the GPCC title bar to access the multi-cluster functionality.

Editing the Multi-Cluster Configuration File

The `clusters.conf` template resides in the `instances/`*`instance_name`*`/conf` directory on each of the nodes in your cluster. Locate this template on the cluster you want to identify as the master cluster; that is 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
```

For example:

```
Miracle:www.miracle.com:28080:Production:True:false
Deforest:10.81.17.186:28080:Development:False:false
Grandalpha:grandalpha:32020:Development:False:false
```

All fields are required.

SERVER

The server value is a primary key, used to uniquely identify each GPCC cluster. This is the display name in the UI. An error message is thrown if there are any duplicate entries in this column. The name may not contain special characters, other than the space character, underscore (_), or hyphen (`-`).

HOST

This is the GPCC hostname or IP address to contact to get health information.

PORT

The port number on which the GPCC is running.

TABGROUP

This field is used to categorize GPCC clusters; for example, *Production*, *Testing*, and *Deployment*.

AUTOLOGIN

This fields enables the auto login feature for the GPCC cluster from the multi-cluster rollup view page.

It takes a true or false value. The value is not case sensitive. Any other value is an error, which will be shown in the UI.

SSL

This field indicates whether SSL is enabled for the host.

It takes a true or false value. The value 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.

Upgrading Greenplum Command Center

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

Upgrading Greenplum Command Center requires stopping the Command Center instance, installing the new distribution, and then recreating Command Center instances.

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 --setup` 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. Stop the Greenplum Command Center service:

   ```
   $ gpcmdr --stop
   ```

4. Download the latest Command Center release from *Pivotal Network*. Installer files are available for the Red Hat 64-bit platform, and have a name in the format:

   ```
   greenplum-cc-web-versionx.x-PLATFORM.zip
   ```

5. Unzip the installer file. For example:

   ```
   # unzip greenplum-cc-web-versionx.x-PLATFORM.zip
   ```

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

   ```
   $ /bin/bash greenplum-cc-web-versionx.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.
7. Read through the license agreement. When you reach the bottom, type `yes` to accept the license agreement.
8. The installer prompts you to provide an installation path. Enter a full path or press ENTER to accept the default. Choose the parent directory of the current Command Center release, `/usr/local` by default. You must have write permission in the directory you specify.
9. 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
   ```

10. Update the `GPPERFMONHOME` environment variable in your user startup shell profile (for example `~/.bashrc`) to point to the newly installed Command Center version:

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

11. Source the updated `.bashrc` file:

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

12. Create a host file listing all of the other hostnames participating in the Greenplum Database cluster, including the Standby Master host. Hostnames must be resolvable in DNS.
13. As `gpadmin`, run the `gpccinstall` utility to install the new Command Center files on all hosts listed in the host file you created:

    ```
    $ gpccinstall -f hostfilename
    ```

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

Recreate Command Center Instances

After the new Command Center software is installed, recreate your instances by running the `gpcmdr --setup` command once for each existing Command Center instance.

See *Set up a Greenplum Command Center Instance* for instructions to run `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
   ```

Chapter 3

Administering Greenplum Command Center

System administration information for the Greenplum Command Center.

- *Starting and Stopping Greenplum Command Center*
- *Configuring Greenplum Command Center*
- *Administering Command Center Agents*
- *Administering the Command Center Database*
- *Web Server Administration*

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"]
```

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`).

- `$GPPERFMONHOME/instances/instance_name/conf/gpperfmonui.conf`
- `$GPPERFMONHOME/instances/instance_name/conf/lighttpd.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.

Changing the gpmon Password

Greenplum Command Center manages the `gpperfmon` database using the `gpmon` database role. 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 PASSWORD 'new_password';
   ```

2. Update the password in the `.pgpass` file, which is used by Greenplum Command Center. The default location for this file is the `gpadmin` home directory (`~/.pgpass`). The `.pgpass` file contains a line with the `gpmon` password. Replace the existing password with the new password.

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

   The file should be owned by `gpadmin` and RW-accessible by `gpadmin` only.
3. Restart Greenplum Command Center with the `gpcmdr` utility.

   ```
   $ gpcmdr --restart
   ```

Administering Command Center Agents

This section 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 described in the Greenplum Command Center Console online help. 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 (`gp_dump`, `gp_restore`, `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.

Web Server Administration

The Lighttpd web server and web service middleware are installed in the `www` directory of your Greenplum Command Center installation. For detailed information on Lighttpd administration, see *http://www.lighttpd.net/*.

Configuring the Web Server

The Lighttpd web server configuration file is stored in `$GPPERFMONHOME/instances/instance_name/conf/lighttpd.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.

You should not need to manually edit this file, if you do, you may break some functionality. Contact Support if you want to make custom modifications to this file.

Viewing and Maintaining Web Server Log Files

Web server access and error logs are written to `$GPPERFMONHOME/instances/instance_name/logs`. These two logs are:

- `lighttpd-access.log`
- `lighttpd-error.log`

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

To prevent web server logs from growing to excessive size, you can set up log file rotation using `logrotate` or `cronolog`, both of which are widely used with Lighttpd.

Chapter 4

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]
     | --status [instance_name]
```

--setup	Configures console components on the installation host. With this option, `gpcmdr` prompts for values to configure the components and writes the values to `gpperfmonui.conf` and `lighttpd.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.
--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 `gpperfmon` utility and the `lighttpd` web service.
--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`, and `--status` you can specify a console instance name. If you do not specify a name, the action applies to all existing console instances.

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
```

Chapter 5

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/conf/gpperfmonui.conf` and **`lighttpd.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.

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

- *Command Center Agent Parameters*
- *Command Center Console Parameters*
- *Greenplum Database Server Configuration Parameters*

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

These parameters only apply to Greenplum Data Computing Appliance platforms.

Each instance of the Command Center Console has two configuration files located in `$GPPERFMONHOME/instances/`*`instance_name`*`/conf`. The web application file is `gpperfmonui.conf` and the web server file is `lighttpd.conf`.

After editing these files, reload the configuration by restarting the Command Center Console instance (`gpperfmon --restart "`*`instance_name`*`"`).

Web Application Parameters

(`gpperfmonui.conf`)

Specifies the instance name displayed on the login page of the Greenplum Command Center Console. This value can be any name you want to display to users, expressed as a text string. Defaults to the instance name you specified when setting up the Command Center Console.

server_name

Specifies the instance name displayed on the login page of the Greenplum Command Center Console. This value can be any name you want to display to users, expressed as a text string. Defaults to the instance name you specified when setting up the Command Center Console.

master_port

Specifies the port number of the Greenplum Database master that this instance is monitoring.

allowautologin

diskthresholdvalue

default 80

securedbhealth

default False

maxconnections

default 10

ssl_enabled

csrf_protect

True/False

timeout

default 1800

pollinterval

30000

ssh_full_path

default ssh

Web Server Parameters

(`lighttpd.conf`)

The lighttpd configuration file has many parameters to configure the web server. The following parameters are the most likely to change for a Command Center Console installation. See the lighttpd documentation for information about other parameters.

server.port

Sets the web server port number. The default HTTP port is 28080.

ssl.engine

Determines whether SSL encryption is used for client connections. Valid values are enable and disable. If you enable SSL at installation time, this is set to enable.

ssl.pemfile

Specifies the path to the PEM file for SSL support. If you enable SSL at installation time, this parameter points to a self-signed certificate. If you use a trusted signed certificate, you must specify it with this parameter.

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.

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_hostname

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.

master_port

The Greenplum Database master port. Default: 5432.

install_wlm

If True, Greenplum Workload Manager is installed. Default: False.

web_port

The listen port for the Command Center lighttpd 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_user_dhe

Set to True to specify the location of an existing Diffie-Hellman parameters (dhparam) file. If False, `gpcmdr` creates a new dhparam file. The file is in PEM format.

enable_reuse_dhe

When set to True, if `gpcmdr` finds an existing `dhparam.pem` file, it will use the existing file instead of creating a new one.

dhe_parm_file

The full path to the dheparam file to use for the instance.

enable_ipv6

Set to True to enable IPV6 connections. Default: False.

enable_csrf_protect

Set to True to enable cross-site request forgery. Default: False.

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.

Example

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
enable_ipv6: false
enable_csrf_protect: true
enable_copy_standby: true
standby_master_host: smdw
enable_ssl: true
enable_user_cert: true
ssl_cert_file: /etc/ssl/certs/cert.pem
enable_user_dhe: false
enable_reuse_dhe: true

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

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

Chapter 6

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.

Column	Type	Description
`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.

Column	Type	Description
`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.

Column	Type	Description
`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.

Column	Type	Description
`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.
`primary_write_shmem_bytes_avg`	bigint	The average amount of data written to shared 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.

Column	Type	Description
`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.

Column	Type	Description
`primary_roundtrip_fsync_msg_microsec_max`	bigint	The maximum 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.
`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.

Column	Type	Description
`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.

Column	Type	Description
`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.

Column	Type	Description
`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.

Column	Type	Description
`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.

Column	Type	Description
`hostname`	varchar(64)	Segment hostname.
`ntype`	varchar(64)	The iterator operation type. Possible values are listed in *Iterator Metrics*.
`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.

Column	Type	Description
`m1_name`	varchar(64)	Each operation in a query plan (`ntype`) has metrics associated with it. See *Iterator Metrics* 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* 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* 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* 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* 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* 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* 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* for a complete list of iterator attributes and their corresponding units.
`m4_val`	bigint	The value of this metric.

Column	Type	Description
`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* 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* 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* 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* 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* 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* 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* for a complete list of iterator attributes and their corresponding units.

Column	Type	Description
`m8_unit`	varchar(64)	The unit of measure for this metric. See *Iterator Metrics* 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* for a complete list of iterator attributes and their corresponding units.
`m9_unit`	varchar(64)	The unit of measure for this metric. See *Iterator Metrics* 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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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

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

Metric	Metric Column	Unit	Description
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.

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

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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.

Metric	Metric Column	Unit	Description
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
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, all having the same columns:

- `log_alert_now` is an external table whose data files are stored in `$MASTER_DATA_DIRECTORY/gpperfmon/data`. Current `pg_log` errors and warnings data is stored in `log_alert_now` during the period between data collection from the Command Center agents and automatic commitment to the `log_alert_history` table.
- `log_alert_tail` is an external table whose data files are stored in `$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

Column	Type	Description
`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

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.

Column	Type	Description
`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.

Column	Type	Description
`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.

Column	Type	Description
`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).

Column	Type	Description
`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.

Column	Type	Description
`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.

Column	Type	Description
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.

Column	Type	Description
`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.
`cpu_currpct`	double precision	The current average percentage of CPU 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.

Column	Type	Description
`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.

Column	Type	Description
`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.