Pivotal

PRODUCT DOCUMENTATION

Pivotal HAWQ

Version 1.2

Installation and Upgrade Guide

Rev: A02 – April 30, 2014

Copyright

Contents

Chapter 1 Preparing to Install HAWQ

This document describes how you can install HAWQ manually. HAWQ can be installed along with Pivotal HD Enterprise using the Command-Line Interface (CLI). However, if you choose to not install HAWQ using the CLI, then you need to follow the instructions in this chapter.

Topics:

Note:

This document does not describe how to install the Pivotal Extension Framework (PXF), which enables SQL querying on data in the Hadoop components such as HBase, Hive, and any other distributed data file types. To install Pivotal Extension Framework (PXF), see the *Pivotal Extension Framework Installation and User Guide*.

Pre-installation

To install HAWQ manually, check that you have met the following prerequisites:

- Review the System Requirements
- Prepare HDFS

Note:

These tasks should be performed for *all* hosts in your HAWQ array (master, standby master and segments).

System Requirements

Check that you meet the following system requirements before you install HAWQ:

Operating System:

- RedHat 6.4 and 6.2, 64 bit
- CentOS 6.4 and 6.2, 64 bit

Minimum CPU: Pentium Pro compatible (P3/Athlon and above)

Minimum Memory: 16 GB RAM per server

Disk Requirements:

- 150MB per host for Greenplum installation
- Approximately 300MB per segment instance for meta data
- Appropriate free space for data: disks should have at least 30% free space (no more than 70% capacity)
- High-speed, local storage

Network Requirements:

- Gigabit Ethernet within the array
- Dedicated, non-blocking switch

Software and Utilities: bash shell, GNU tar, and GNU zip

Preparing HDFS

You need to complete the following steps to configure HDFS:

1. Download the HDFS binaries. See the *Pivotal HD Enterprise Installation and Administrator Guide* for more information.
2. Install the rpms. See the *Pivotal HD Enterprise Installation and Administrator Guide* for more information.
3. To make sure HDFS block files can be read by other users, configure OS file system umask to 022.
4. Start up the HDFS service. See the *Pivotal HD Stack and Tool Reference Guide* for more information.
5. Add the following line into ${HADOOP_HOME}/etc/hadoop/hadoop-env.sh

   ```
   umask 022
   ```

6. Add the following parameter to hdfs-site.xml

   ```
   <property>
       <name>dfs.datanode.data.dir.perm</name>
       <value>755</value>
       <description>Permissions for the directories on on the local filesystem where the DFS data
   node store its blocks. The permissions can either be octal or symbolic.</description>
   </property>
   ```

7. Edit the */hdfs-install-directory/etc/hadoop/hdfs-site.xml* file. To do this, change the *dfs.block.local-path-access.user* to the user who starts HDFS if the short circuit feature is enabled in libhdfs3. See the *HAWQ Configuration Parameter Reference* for more information.
8. Set the *dfs.namenode.name.dir* and *dfs.datanode.data.dir* to your preferred path as shown in the example:

   ```
   <configuration>
      <property>
         <name>dfs.support.append</name>
         <value>true</value>
      </property>
      <property>
          <name>dfs.client.read.shortcircuit</name>
          <value>true</value>
      </property>
      <property>
          <name>dfs.block.local-path-access.user</name>
          <value>gpadmin</value>
          <description>
                specify the user allowed to do short circuit read
          </description >
      </property>
      <property>
   ```

```
      <name>dfs.namenode.name.dir</name>
<value>file:/home/gpadmin/hadoop-2.0.0-alpha/dfs/name</value>
</property>
    property>
      <name>dfs.datanode.data.dir</name>
      <value>file:/home/gpadmin/hadoop-2.0.0-alpha/dfs/data</value>
   </property>
   <property>
      <name>dfs.replication</name>
      <value>3</value>
   </property>
   <property>
      <name>dfs.datanode.max.transfer.threads</name>
      <value>40960</value>
   </property>
   <property>
</property>
   <property>
      <name>dfs.client.socket-timeout</name>
      <value>300000000</value>
   </property>
   <property>
      <name>dfs.datanode.handler.count</name>
      <value>60</value>
   </property>
<property>
<name>ipc.client.connection.maxidletime</name>
<value>3600000</value>

</property>
<property>
<name>ipc.server.handler.queue.size</name>
<value>3300</value>

</property>

</property>
   <property>
<name>ipc.client.connection</name>
      <value>3</value>
   </property>
   <property>
      <name>dfs.datanode.max.transfer.threads</name>
      <value>40960</value>
   </property>
   <property>
</property>
   <property>
      <name>dfs.replication</name>
      <value>3</value>
   </property>
   <property>
      <name>dfs.namenode.accesstime.precision</name>
      <value>-1</value>
   </property>
   <property>
```

9. To configure the JVM, edit the */hdfs-install-directory/etc/hadoop/hadoop-env.sh*

   This configures the memory usage of the primary and secondary namenodes and datanode. For example, on servers with 48GB memory, if HDFS and HAWQ are on two separate clusters, Pivotal recommends that the namenodes use 40GB (-Xmx40960m), while each datanode uses 6GB and with a stack size of 256KB (-Xmx6144m -Xss256k).

10. To verify that HDFS has started, run the following command sequence.

    a. List the directory:

    ```
    hadoop fs -ls /
    ```

    b. Create a test directory:

    ```
    hadoop fs -mkdir /test
    ```

    c. Put a test file *(/path/file*) into the HDFS root directory:

    ```
    hadoop fs -put /path/file /
    ```

    d. Perform a get on */file* from HDFS to the current local file system directory:

    ```
    hadoop fs -get /file ./
    ```

Chapter 2 Installing HAWQ

This section contains procedures to help you install HAWQ.

Topics:

Install the HAWQ Binaries

You can install HAWQ from a RPM or binary tarball release.

To Install the RPM Release

1. Log in to the master host as *root*.

```
$ su - root
```

2. Launch the installer using rpm. For example:

```
# rpm –ivh hawq-dev-dev.x86_64.rpm
```

   The installer installs HAWQ to the default install path (*/usr/local/hawq-dev*), and creates the soft link */usr/local/hawq* for */usr/local/hawq-dev*.

3. Source the path file from your master host's HAWQ installation directory:

```
# source /usr/local/hawq/greenplum_path.sh
```

4. Create a file called *hostfile* that includes host names in your HAWQ system using segment hosts. Make sure there are no blank lines or extra spaces. For example, if you have a standby master and three segments per host, your file will look something like this:

```
smdw
sdw1
sdw2
sdw3
```

5. Perform the ssh key exchange by running the following command. This allows you to log in to all hosts as root user without a password prompt. Use the *hostfile* file you used for installation.

```
gpssh-exkeys -f hostfile
```

6. Run the following command to reference the *hostfile* file you just created and copy the HAWQ rpm file (*hawq-dev-dev.x86_64.rpm*) to all hosts:

```
gpscp -f hostfile
hawq-dev-dev.x86_64.rpm =:~/
```

7. Run the following command to install HAWQ to all hosts:

```
# gpssh -f hostfile -e "rpm -ivh hawq-dev-dev.x86_64.rpm"
```

To Install from a Binary Tarball

1. Log in to the master host as root.

```
# su - root
```

2. Copy the HAWQ tarball to the binary directory you want to install HAWQ, go to the binary directory and uncompress the tarball. For example:

```
# cp /path/to/hawq-dev-dev.tar.gz /usr/local
# cd /usr/local
# tar xf hawq-dev-dev.tar.gz
```

   A HAWQ directory is generated.

3. Open the file */usr/local/greenplum_path.sh* and edit the *GPHOME* parameter to set it to */usr/local/hawq .*

```
GPHOME=/usr/local/hawq
```

4. Source the path file from your master host's HAWQ installation directory:

```
# source /usr/local/hawq/greenplum_path.sh
```

   Create a file called *hosttfile* that includes host names used in your HAWQ system in segment hosts format. Make sure there are no blank lines or extra spaces. For example, if you have a standby mnaster and three segments per host, your file will look something like this:

```
smdw
sdw1
sdw2
sdw3
```

5. Perform the *ssh* key exchange by running the following command. This allows you to log in to *all_hosts* as root *user* without a password prompt. Use the *all_hosts* file you used for installation:

```
# gpssh-exkeys -f all_hosts
```

6. Run the following commands to reference the hostfile file you just created and copy the HAWQ binary directory (*/usr/local/hawq-dev*) to all hosts:

```
# gpscp -r -f hostfile hawq-dev =:/usr/local/
# gpssh -f hostfile -e "ln -s  /usr/local/hawq-dev /usr/local/hawq"
```

Creating the gpadmin User

1. Create the *gpadmin* user account on each host:

```
# gpssh -f all_hosts -e '/usr/sbin/useradd gpadmin'
# gpssh –f all_hosts -e 'echo -e "changeme\nchangeme" | passwd gpadmin'
```

2. Log in to the master host as *gpadmin*:

```
$ su - gpadmin
```

3. Source the path file from the HAWQ installation directory:

```
$ source /usr/local/hawq/greenplum_path.sh
```

4. Run the following command to do the *ssh* key exchange to enable you to log in to all hosts without a password prompt as *gpadmin* user. Use the *all_hosts* file you used for installation:

```
$ gpssh-exkeys -f all_hosts
```

5. Use the *gpssh* utility to add the above command line to the profile file. For example:

```
$ gpssh -f all_hosts -e "echo source /usr/local/ hawq/greenplum_path.sh >> .bashrc"
```

6. Use the *gpssh* utility to confirm that the Pivotal software was installed on all hosts. Use the *all_hosts* file you used for installation. For example:

```
$ gpssh -f all_hosts -e "ls -l $GPHOME"
```

> **Note:**
>
> You may want to change the default configuration parameters in */usr/local/hawq/etc/hdfs-client.xml* for *libhdfs3*. See the topic, HAWQ Configuration Parameter Reference.

7. Log in to the master host as *root*:

```
$ su - root
```

Setting the OS Parameters

This topic describes the OS parameter options that you need to set up for the following:

- Linux
- RHEL
- Security Configuration
- XFS

Linux

Note:

Pivotal recommends that you do not set the *vm.overcommit_memory parameter* if you run HAWQ on small memory virtual machines. If you set this parameter you may encounter out of memory issues.

Set the following parameters in the */etc/sysctl.conf* file and reboot:

```
sysctl.kernel.shmmax = 500000000
sysctl.kernel.shmmni = 4096
sysctl.kernel.shmall = 4000000000
sysctl.kernel.sem = 250 512000 100 2048
sysctl.kernel.sysrq = 1
sysctl.kernel.core_uses_pid = 1
sysctl.kernel.msgmnb = 65536
sysctl.kernel.msgmax = 65536
sysctl.kernel.msgmni = 2048
sysctl.net.ipv4.tcp_syncookies = 0
sysctl.net.ipv4.ip_forward = 0
sysctl.net.ipv4.conf.default.accept_source_route = 0
sysctl.net.ipv4.tcp_tw_recycle = 1
sysctl.net.ipv4.tcp_max_syn_backlog = 200000
sysctl.net.ipv4.conf.all.arp_filter = 1
sysctl.net.ipv4.ip_local_port_range = 1025 65535
sysctl.net.core.netdev_max_backlog = 200000
sysctl.vm.overcommit_memory = 2
sysctl.fs.nr_open = 3000000
sysctl.kernel.threads-max = 798720
sysctl.kernel.pid_max = 798720
#increase network
sysctl.net.core.rmem_max = 2097152
sysctl.net.core.wmen_max = 2097152
```

RHEL

For RHEL version 6.x platforms, the above parameters do not include the *sysctl.* prefix, as follows:

```
kernel.shmmax = 500000000
kernel.shmmni = 4096
kernel.shmall = 4000000000
kernel.sem = 250 512000 100 2048
kernel.sysrq = 1
kernel.core_uses_pid = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.msgmni = 2048
net.ipv4.tcp_syncookies = 0
net.ipv4.ip_forward = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_max_syn_backlog = 200000
net.ipv4.conf.all.arp_filter = 1
net.ipv4.ip_local_port_range = 1025 65535
net.core.netdev_max_backlog = 200000
vm.overcommit_memory = 2
fs.nr_open = 3000000
kernel.threads-max = 798720
kernel.pid_max = 798720
# increase network
net.core.rmem_max=2097152
net.core.wmem_max=2097152
```

Security Configuration

After updating the */etc/sysctl.conf* file, set the following parameters (in the exact sequence displayed in the example) in the */etc/security/limits.conf* file:

```
soft nofile 2900000
hard nofile 2900000
soft nproc 131072
hard nproc 131072
```

XFS

XFS is the preferred file system on Linux platforms for data storage. Pivotal recommends the following xfs mount options:

```
rw,noatime,inode64,allocsize=16m
```

You need to change the allocsize to 64k, only in the case of the master and the standby. To do so, change the allocsize to 64k in the /etc/fstab file. Run the following commands:

```
sudo umount -l /path/to/filesystem
sudo mount /path/to/filesystem
```

See the Linux manual page (man) for more information about the mount command:

The Linux disk I/O scheduler for disk access supports different policies, such as CFQ, AS, and deadline.

Pivotal recommends the following scheduler option:

To specify a scheduler, run the following:

```
# echo schedulername > /sys/block/devname/queue/scheduler
```

For example:

```
# echo deadline > /sys/block/sbd/queue/scheduler
```

Each disk device file should have a read-ahead (blockdev) value of 16384. To verify the read-ahead value of a disk device:

```
# /sbin/blockdev --getra devname
```

For example:

```
# /sbin/blockdev --getra /dev/sdb
```

To set blockdev (read-ahead) on a device:

```
# /sbin/blockdev --setra bytes devname
```

For example:

```
# /sbin/blockdev --setra 16385 /dev/sdb
```

Refer to the Linux manual page (man) for more information about using the blockdev command.

Editing the Configuration Files

Edit the /etc/hosts file and make sure that it includes the host names and all interface address names for every machine participating in your HAWQ system.

1. Run the following command to copy the /etc/sysctl.conf file and /etc/security/limits.conf file to the same location of all hosts:

```
# gpscp -f all_hosts /etc/sysctl.conf =:/etc
# gpscp -f all_hosts /etc/security/limits.conf =:/etc/security
```

Note:

You may need to configure other parameters(for example, scheduler configuration)on all hosts.

2. Create or choose a directory that will serve as your master data storage area. This directory should have sufficient disk space for your data and be owned by the gpadmin user and group. For example, run the following commands as root:

```
# mkdir /data/master
```

3. Change ownership of this directory to the gpadmin user. For example:

```
# chown -R gpadmin /data/master
```

4. Using gpssh, create the master data directory location on your standby master as well. For example:

```
# gpssh -h smdw -e 'mkdir /data/master'
# gpssh -h smdw -e 'chown -R gpadmin /data/master'
```

5. Create a file called seg_hosts. This file should have only one machine configured host name for each segment host. For example, if you have three segment hosts:

```
sdw1
sdw2
sdw3
```

6. Using gpssh, create the data directory locations on all segment hosts at once using the seg_hosts file you just created. For example:

```
# gpssh -f seg_hosts -e 'mkdir /data/primary'
# gpssh -f seg_hosts -e 'chown gpadmin /data/primary'
```

7. To use JBOD, create temporary directory locations for the master, standby, and all the segments. The following example uses two disks with the workfile names /data1/tmp and /data2/tmp.

```
# dirs="/data1/tmp /data2/tmp"
# mkdir $dirs# chown -R gpadmin $dirs.
# gpssh -h smdw -e "mkdir $dirs"
# gpssh -h smdw -e "chown -R gpadmin $dirs"
# gpssh -f seg_hosts -e "mkdir $dirs"
# gpssh -f seg_hosts -e "chown -R gpadmin $dirs"
```

8. Log in to the master host as gpadmin. Make a copy of the gpinitsystem_config file to use as a starting point. For example:

```
$ su - gpadmin$ cp
$GPHOME/docs/cli_help/gpconfigs/gpinitsystem_config /home/gpadmin/gpconfigs/gpinitsystem_config
```

9. Open the file you just copied in a text editor. Set all of the required parameters according to your environment. A HAWQ system must contain a master instance and at least two segment instances (even if setting up a single node system). The DATA_DIRECTORY parameter is what determines how many segments per host will be created. Here is an example of the required parameters in the gpinitsystem_config file:

```
ARRAY_NAME="EMC GP-SQL"
SEG_PREFIX=gpseg
PORT_BASE=40000
declare -a TEMP_DIRECTORY=(/data1/tmp /data2/tmp)
declare -a DATA_DIRECTORY=(/data/primary /data/primary)
MASTER_HOSTNAME=mdw
MASTER_DIRECTORY=/data/master
MASTER_PORT=5432
TRUSTED SHELL=ssh
CHECK_POINT_SEGMENT=8
ENCODING=UNICODE
DFS_NAME=hdfs
DFS_URL=mdw:9000/gpsql
```

Ensuring that HDFS works

1. Make sure that your hdfs is working and change the following parameters in the gpinitsystem_config:

```
DFS_NAME=hdfs
DFS_URL=namenode-host-name:8020/hawq
```

2. Save and close the file.

3. Run the following command referencing the path and file name of your initialization configuration file (gpinitsystem_config) and host file (seg_hosts). For example:

```
$ cd ~
$ gpinitsystem -c gpconfigs/gpinitsystem_config -h seg_hosts
```

For a fully redundant system (with a standby master and a spread mirror configuration) include the -s and -S options. For example:

```
$ gpinitsystem -c gpconfigs/gpinitsystem_config -h seg_hosts -s standby_master_hostname
```

The utility verifies your setup information and ensures that it can connect to each host and access the data directories specified in your configuration. If all of the pre-checks are successful, the utility prompts you to confirm your configuration. For example:

```
=> Continue with Greenplum creation? Yy/Nn
Press y to start the initialization.
```

The utility begins setup and initialization of the master and each segment instance in the system. Each segment instance is set up in parallel. Depending on the number of segments, this process can take a while.

4. Set the MASTER_DATA_DIRECTORY environment variable. For example, add the following line to the profile of the master host:

```
export MASTER_DATA_DIRECTORY=/data/master/gpseg-1
```

HAWQ on Secure HDFS

Requirements

- A secure HDFS installation
- HDFS on wire encryption (`dfs.encrypt.data.transfer`) MUST be set to `false`.
- A new un-initialized HAWQ instance or a stopped already initialized HAWQ instance that was previously running on non-secured HDFS

Preparation

a. If HAWQ is already initialized and running, stop HAWQ by running `service hawq stop` or `<HAWQ installation directory>/bin/gpstop`.

b. Secure the HDFS cluster using the instructions provided in the *Pivotal HD Stack and Tool Reference Guide* or using available security tools.

c. Insure HDFS is running properly in secured mode.

d. Insure that the property `dfs.encrypt.data.transfer` is set to `false` in the `hdfs-site.xml` for your cluster.

Configuration

a. Generate a "postgres" principal and keytab file as shown below:

The form of principal for the HAWQ master is `postgres@REALM`, where `postgres` is the default service name of HAWQ and `REALM` is the default realm in the cluster's Kerberos configuration. In the examples below we use `EXAMPLE.COM` for the `REALM` part; this should be replaced by your cluster's actual `REALM`.

```
kadmin: addprinc -randkey postgres@EXAMPLE.COM
kadmin: ktadd -k /etc/security/phd/keytab/hawq.service.keytab postgres@EXAMPLE.COM
```

b. Move this keytab file to the appropriate keytab directory on the HAWQ master node (for example, `/etc/security/phd/keytab/`).

c. Set the ownership of the keytab file to `gpadmin:gpadmin` and the permissions to 400.

d. Refer to your `gpinitsystem_config` file (typically in `/etc/gphd/hawq/conf`) to determine your configured HAWQ HDFS data directory (typically `/hawq_data`). This will be the last part of the `DFS_URL` value. For example if `DFS_URL` is set to `centos61-2:8020/hawq_data` then your HAWQ HDFS data directory is `/hawq_data`.

e. Create (if required) the HAWQ HDFS data directory in HDFS, and assign ownership as `postgres:gpadmin` and permissions 755.

- If HAWQ has already been initialized and the directory exists just modify the owner and permissions as shown.
- You need to have HDFS super-user permissions to create or modify a directory in HDFS root. If necessary create an "hdfs" principal to accomplish this task.

f. Create in HDFS (if not present) the directory `/user/gpadmin` with ownership `gpadmin:gpadmin` and permissions 777.

g. Modify the `hdfs-client.xml` file (typically in `/usr/lib/gphd/hawq/etc`) on the master node and ALL segment server nodes by adding the following:

```
<property>
  <name>hadoop.security.authentication</name>
  <value>kerberos</value>
</property>

<property>
  <name>dfs.namenode.kerberos.principal</name>
  <value>HDFS_NAMENODE_PRINCIPAL</value>
</property>
```

- `hdfs-client.xml` is in `<HAWQ installation directory>/etc`, typically `/usr/lib/gphd/hawq/etc`.
- These property blocks should be in the file but commented out, if so uncomment and edit the values.
- `HDFS_NAMENODE_PRINCIPAL` should be value from your cluster's `hdfs-site.xml` file.
- Make sure the namenode principal value is correct.

h. Edit your `gpinitsystem_config` file (typically in `/etc/gphd/hawq/conf`) and add (or uncomment if they are present and commented out):

```
KERBEROS_KEYFILE=/path/to/keytab/file
ENABLE_SECURE_FILESYSTEM=on
```

- Make sure there is no space between the `key=value`; for example: `ENABLE_SECURE_FILESYSTEM = on` will cause errors because there are spaces.
- Make sure the value of `KERBEROS_KEYFILE` is the full path of where you placed the `hawq.service.keytab` file on the master.

i. If HAWQ has already been initialized prior to being secured run the following commands on the HAWQ master as the `gpadmin` user:

   i. `service hawq start`

   ii. `source /usr/local/hawq/greenplum_path.sh`

   iii. `gpconfig --masteronly -c krb_server_keyfile -v "'/path/to/keytab/file'"`
   NOTE The single quotes ' after/before the double quotes " in the keytab string above are required!

   iv. `service hawq stop`

j. After you have completed all these steps you can start or initialize HAWQ:

   i. If HAWQ was already initialized on non-secured HDFS before this process, start it by running `service hawq start` or `<HAWQ installation directory>/bin/gpstart`.

   ii. If HAWQ has not been initialized, initialize it now.

k. Verify HAWQ is operating properly, if not see next section.

Troubleshooting

If initialization or start-up fails you can look into the gpinitsystem log output and the namenode logs to see if you can pinpoint the cause. Possible causes:

- Incorrect values in your `hdfs-client.xml`
- `hdfs-client.xml` not updated on master and all segment servers
- Unable to login with Kerberos; possible bad keytab or principal for "postgres"
  - Validate on master by doing: `kinit -k <keytab dir path>/hawq.service.keytab postgres@EXAMPLE.COM`
- Wrong HAWQ HDFS data directory or directory permissions: Check your `gpinitsystem_config` file and the `DFS_URL` value and the directory permissions.
- Unable to create the HAWQ HDFS data directory errors: insure that you have created the proper directory as specified in `gpinitsystem_config` and that the ownership and permissions are correct.

Creating a HAWQ Instance on HDFS with Namenode High Availability (HA)

Before you proceed, check that HDFS is configured with the Namenode HA feature.

1. Edit the ${GPHOME}/etc/hdfs-client.xml file:

```
<property>
    <name>dfs.nameservices</name>
    <value>phdcluster</value>
</property>
<property>
    <name>dfs.ha.namenodes.phdcluster</name>
    <value>nn1,nn2</value>
</property>
<property>
    <name>dfs.namenode.rpc-address.phdcluster.nn1</name>
    <value>mdw:9000</value>
</property>
<property>
    <name>dfs.namenode.rpc-address.phdcluster.nn2</name>
    <value>smdw:9000</value>
</property>
<property>
    <name>dfs.namenode.http-address.phdcluster.nn1</name>
    <value>mdw:50070</value>
</property>
<property>
    <name>dfs.namenode.http-address.phdcluster.nn2</name>
    <value>smdw:50070</value>
</property>
<property>
    <name>dfs.client.failover.proxy.provider.phdcluster</name>
    <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
</property>
```

Notes

Change this file on the HAWQ master and all segments.

Replace the phdcluster to real service ID configured in HDFS.

Replace mdw:9000 and smdw:9000 to real namenode RPC host and port configured in HDFS.

Replace mdw:50070 and smdw:50070 to real namenode HTTP host and port configured in HDFS.

The namenodes order in the value of "dfs.ha.namenodes.phdcluster" is important to the performance, especially when running on security enabled HDFS.

```
<property>
    <name>dfs.ha.namenodes.phdcluster</name>
    <value>nn1,nn2</value>
</property>
```

Please make sure nn1 is the active namenode to prevent a failover cost for HAWQ. To check that the active namenode is nn1 reorder the values for "dfs.ha.namenodes.phdcluster".

If this parameter is changed, please make sure it is changed on all nodes.

2. To prepare the configuration file for the command line tool, gpinitsystem, change the following parameters in the gpinitsystem_config file:

```
DFS_NAME=hdfs
DFS_URL=phdcluster/path/to/hawq/data
```

Note

- Replace phdcluster to real service ID configured in HDFS.
- Replace /path/to/hawq/data to the directory where the user want to store the data on HDFS, and make sure it exists and is writable.

Running a Basic Query

You can run the create database query to test that HAWQ is running:

```
changl1-mbp:gpsql changl1$ psql -d postgres
psql (8.2.15)
Type "help" for help.
postgres=# create database tpch;
CREATE DATABASE
postgres=# \c tpch
You are now connected to database "tpch" as user "changl1".
tpch=# create table t (i int);
NOTICE:  Table doesn't have 'DISTRIBUTED BY' clause -- Using column named 'i' as the Greenplum
Database data distribution key for this table.
HINT:  The 'DISTRIBUTED BY' clause determines the distribution of data. Make sure column(s) chosen
are the optimal data distribution key to minimize skew.
CREATE TABLE
tpch=# \timing
Timing is on.
tpch=# insert into t select generate_series(1,100);
INSERT 0 100
Time: 311.390 ms
tpch=# select count(*) from t;
count
-------
100
(1 row)
Time: 7.266 ms
```

Chapter 3 Installing the HAWQ Components

This chapter describes how to install additional HAWQ components.

Topics:

Installing Cryptographic Functions for PostgreSQL

The pgcrypto package contains the cryptographic functions for PostgreSQL. It must be installed separately from the main installation.

> **Note:**
>
> Your HAWQ installation must be running, or the pgcrypto package will fail to install.

All the cryptographic functions run inside database server. That means that all the data and passwords move between pgcrypto and the client application in clear-text. This means you must:

- Connect locally or use SSL connections.
- Trust both system and database administrator.

The pgcrypto package can be installed either through automated or manual installation.

> **Note**
>
> Download the new binaries for the pgcrypto package and follow the appropriate instructions in this topic.

Automated Installation of pgcrypto

These installation instructions assume you have obtained a precompiled build of pgcrypto from Pivotal.

1. Extract files from the pgcrypto package. For example:

```
mkdir pgcrypto
mv pgcrypto.tgz pgcrypto
tar -xzvf pgcryto/pgcrypto.tgz
```

2. Run pgcrypto_install.sh script with the hostfile as argument. The hosts file must contain hostname of each segment, one per line.

```
cd pgcrypto
./pgcrypto_install.sh -f ~/hostfile
```

3. Install the pgcrypto function:

```
psql -d <dbname> -f $GPHOME/share/postgresql/contrib/pgcrypto.sql
```

Uninstalling pgcrypto

To uninstall the pgcrypto objects, use uninstall_pgcrypto.sql.

Note:

This script does not remove dependent user created objects.

Installing PL/R

The following instructions assume that you have downloaded a precompiled build of PL/R from Pivotal.

Note:

Check the name of your plr package. If it is plr-1.1.4.0-5152.x86_64.tgz,download the latest version plr-1.1.4.0-5664.x86_64.tgz for HAWQ 1.1.4.0 from Pivotal. The new package contains the file plr.sql with the necessary PL/R helper functions.

1. Create a directory named plr and extract the archive in it.

```
mkdir plr
mv plr*.tgz plr
tar -xzf plr*.tgz -C plr
```

2. Run the plr_install.sh script with the hostfile as argument. The hosts file must contain hostname of each segment, one per line..

```
cd plr;
./plr_install.sh -f ~/hostfile
```

3. After the installation finishes successfully, restart the HAWQ instance.

```
source $GPHOME/greenplum_path.sh
gpstop -ar
```

4. Create the plr language. Connect to a database using PSQL and run the following SQL command.

```
psql -d template1 -c "CREATE LANGUAGE plr"
```

You are now ready to create new PLR functions. A library of convenient PLR functions may be found in $GPHOME/share/postgresql/contrib/plr.sql. These functions may be installed by executing plr.sql, as follows:

```
psql -d <dbname> -f $GPHOME/share/postgresql/contrib/plr.sql
```

Installing PL/Java

Notes

Before you install PL/Java

- You have downloaded a precompiled build of PL/Java from Pivotal.
- Ensure that the $JAVA_HOME variable is set to the same path on the master and all the segments.
- If you are upgrading to the latest version of Java or installing it as part of the expansion process, follow the instructions in the chapter, *Expanding the HAWQ System* in the *HAWQ Administrator Guide*.
- PL/Java is compatible with JDK 1.6 and 1.7.

1. Extract files from the PL/Java package:

```
mkdir pljava
mv pljava*.tgz pljava
cd pljava
tar -xzf pljava.tgz
```

2. Run the installer.

```
./pljava_install.sh -f ~/hosts.txt
```

Where the ~/hosts.txt file contains the names of currently active segment hosts in the HAWQ deployment. You must format the file to list one hostname per line.

3. Restart HAWQ.

```
source $GPHOME/greenplum_path.sh gpstop -ar
```

4. Add the pljava class of configuration variables:

```
gpconfig -c custom_variable_classes -v \'pljava\'
```

To add previously defined custom_variable_classes, prefix them with "pljava" in a comma separated list.

5. Create the language for pljava.

```
psql -d <dbname> -c "CREATE LANGUAGE pljava"
```

Installing Custom JARS

1. Copy the jar file on the master host in $GPHOME/lib/postgresql/java
2. Copy the jar file on all segments in the same location using gpscp from master.

```
cd $GPHOME/lib/postgresql/java gpscp -f ~/hosts.txt myfunc.jar =:$GPHOME/lib/postgresql/java/
```

3. Set the pljava_classpath to include the newly copied jar file.
   a. From the psql session, execute set to affect the current psql session.

```
pljava_classpath='myfunc.jar';
```

   b. To affect all sessions:

```
gpconfig -c pljava_classpath -v \'myfunc.jar\'
```

Installing MADlib on HAWQ

The MADlib library adds statistical and machine learning functionality to HAWQ. This topic describes how to install MADlib for HAWQ.

The HAWQ installation script installs the MADlib files, but does not register MADlib functions with HAWQ databases. Therefore you must use the `madpack` utility program to install, reinstall, or upgrade the MADlib database objects.

Upgrading HAWQ from 1.1 to 1.2

If you have upgraded your system from HAWQ 1.1.x to HAWQ 1.2, you must install MADlib 1.5 or higher and then run the `madpack` utility program.

Pre-requisites for Installing MADlib on HAWQ

Check that you have completed the following tasks before running the installation script:

- Make sure you have `rpm`, `gpssh` and `gpscp` in your `PATH`.
- Make sure that you have HAWQ binaries installed properly on all master and segment nodes in your cluster (also new segment nodes when adding new nodes).
- Add `hawq_install.sh` to your `PATH`.
- Make sure the `HOSTFILE` lists all the new segment nodes.

Installing MADlib on HAWQ

1. Once HAWQ installation is complete, run the following command to install MADlib:

```
hawq_install.sh -r <RPM_FILEPATH> -f <HOSTFILE> [-s] [-d <GPHOME>] [--prefix
<MADLIB_INSTALL_PATH>]
```

2. Run the following command to register MADlib in your database:

```
$GPHOME/madlib/bin/madpack -p hawq -c $USER@$HOST/$DATABASE install
```

3. To test your installation, run the following command:

```
$GPHOME/madlib/bin/madpack -p hawq -c $USER@$HOST/$DATABASE install-check
```

MADlib Install Required Settings

- `-r | --rpm-path <RPM_FILEPATH>` The p ath to the MADlib RPM file.
- `-f | --host-file <HOSTFILE>` The f ile containing the host names of all new segments.

MADlib Install Optional Settings

- `-s | --skip-localhost` Set this option to prevent MADlib installation on the localhost.
- `-d | --set-gphome <GPHOME>` Indicates the HAWQ installation path. If you do not specify one, the installer uses the value stored in the environment variable `GPHOME`.
- `--prefix <MADLIB_INSTALL_PATH>` Indicates MADlib installation path. If not set, will use the default value in MADlib RPM.
- `-h | -? | --help` Displays help.

MADlib Install Example

```
hawq_install.sh -r /home/gpadmin/madlib/madlib-1.5-Linux.rpm -f /usr/local/greenplum-db/hostfile
```

Chapter 4 Upgrading HAWQ and Components

This section describes how to upgrade HAWQ and its components.

Topics:

Upgrading HAWQ

The upgrade path supported for this release is HAWQ 1.1.x to HAWQ 1.2.0.0. Pivotal recommends that you use the ICM to upgrade your HAWQ system from 1.1.x to 1.2.0.0.

Notes

- Pivotal recommends that you back up any existing data before upgrading to HAWQ1.2.0.0
- Follow these instructions if you installed HAWQ manually. To upgrade PHD Manager, see the *Pivotal HD Enterprise 2.0 Installation and Administration Guide*.

Preparing to Upgrade HAWQ

Perform these steps on your current HAWQ system. This procedure is performed from your HAWQ master host and should be executed by the HAWQ superuser (gpadmin).

1. Log in to the HAWQ master as the gpadmin user
2. (*optional*) Vacuum all databases prior to upgrade.
3. (*optional*) Clean out old server log files from your master and segment data directories.

Note

Running Vacuum and cleaning out old logs files is not required, but it will reduce the size of HAWQ files to be backed up and migrated.

4. Run gpstate to check for failed segments. If you have failed segments, you must recover them using gprecoverseg before you can upgrade.
5. Copy or preserve any additional folders or files (such as backup folders) that you have added in the HAWQ data directories or $GPHOME directory. Only files or folders strictly related to HAWQ operations are preserved by the migration utility.

Upgrading to HAWQ 1.2.0.0

An upgrade from HAWQ 1.1.x to HAWQ 1.2.0.0 involves stopping HAWQ, updating the HAWQ software binaries, and restarting HAWQ.

1. Log in to your HAWQ master host as the HAWQ administrative user:

```
$ su - gpadmin
```

2. Perform a smart shutdown of your current HAWQ 1.1.x system (shut down all active connections to the database):

```
$ gpstop
```

3. Run the installer for 1.2.0.0 on the HAWQ master host using rpm. This installs HAWQ to /usr/local/hawq-1.2.0.0 alongside any older versions, and it will point a soft link from /usr/local/hawq to /usr/local/hawq-1.2.0.0

```
$ su - root # rpm -ivh hawq-1.2.0.0.x86_64.rpm --force
```

4. Run the following command to install the HAWQ 1.2.0.0 binaries on all the hosts specified in the *hostfile* :

```
# gpssh -f hostfile -e "rpm -ivh hawq-1.2.0.0.x86_64.rpm --force"
```

5. After all segment hosts have been upgraded, you can log in as gpadmin user and restart your HAWQ system:

```
$ su - gpadmin
$ gpstart
```

Upgrading the Components

This section describes how you can upgrade components such as MADlib, PL/R, pgcrypto, and PL/Java.

If you have previously installed versions of PL/R, and pgcrypto packages, then you must download newer binary compatible version of these packages from Pivotal. You will then need to re-install the newer binaries. See [#Installing Cryptographic Functions for PostgreSQL for instructions.

PL/Java is a new component, therefore if you are installing PL/Java as a part of the upgrade or expansion process then make sure you have carried out steps as described in the chapter, *Expanding your HAWQ System*, in the *HAWQ Administrator Guide*.

Prerequisites for Upgrading MADlib

1. Ensure that you have upgraded PHD 1.1 to PHD 2.0.
2. Ensure that HAWQ 1.2.0.0 is installed on all nodes.

During a HAWQ upgrade to 1.2, the gpmigrator script drops the built-in MADlib objects (functions, aggregates, and types).If you have user defined (UDF) objects that depend on these Madlib objects, then the gpmigrator script detects direct or indirect dependencies in all the databases and creates a log. To proceed with the migration:

1. a. Backup the data.

   b. Drop all UDFs before resuming the migration process.

Upgrading MADlib on HAWQ 1.2.0.0

1. Log in to your HAWQ master host as the HAWQ administrative user:
2. To check for Madlib dependencies, set the environment variables: `PGPORT(default=5432)`, `PGUSER` (default=current logged on user), `PGHOST(default=localhost)` and `PGPASSWORD` (If empty the script will prompt for a password)
3. Run `gpmigrator SOURCE_VERSION_GPHOME TARGET_VERSION_GPHOME`
4. If there are user defined objects that depend on Madlib functionality, then the script will detect these and halt. Take appropriate action to remove these dependencies.

When the gpmigrator reports success the new HAWQ database will be up and running. The gpmigrator script backs up the original catalog and restores it in case of failures. This leaves the database in a usable state.

Troubleshooting a Failed Upgrade

If you experience issues during the migration process, go to the Support page at Support Zone or contact Pivotal customer support at one of the following numbers:

- United States: 800-782-4362 (1-800-SVC-4EMC)
- Canada: 800-543-4782
- Worldwide: +1-508-497-7901

Be prepared to provide the following information:

- A detailed list of upgrade procedures completed.
- Log output from gpmigrator (located in ~/gpAdminLogs).

Chapter 5 HAWQ Configuration Parameter Reference

Describes the configuration in the path $HAWQ_install_path/etc/hdfs-client.xml.

Parameter	Description	Default value	Comments
ipc.client.connection.maxidletime	The idle timeout interval of a rpc channel, rpc channel will exit if timeout occurs.	10000 (ms)	
ipc.client.connect.max.retries	The max retry times when a rpc channel failed to connect to the server for any reason except timeout.	1	
ipc.client.connect.max.retries	The max retry times when a rpc channel failed to connect to the server for any reason except timeout.	1	
ipc.client.connect.max.retries.on.timeouts	The max retry times when a rpc channel failed to connect to the server since timeout.	1	
ipc.client.connect.timeout	The timeout interval for a rpc channel to connect to the server.	10000 (ms)	
ipc.client.write.timeout	The timeout interval for a rpc channel to write data into socket	10000 (ms)	
ipc.client.tcpnodela	To set rpc channel to tcpnodelay mode	true	
dfs.ConfigKey.type	Default checksum type, valid value is: CRC32, CRC32C, NULL	CRC32C	Must be set to the same value as the HDFS side.
dfs.bytes-per-ConfigKey	The size of chunk to calculate checksum.	512	
dfs.datanode.rpc.timeout	The timeout interval to the client to wait for the datanode to finish a rpc call.	3	
dfs.namenode.rpc.timeout	The timeout interval to the client to wait for the namenode to finish a rpc call.	3600000	
dfs.client-write-packet-size	The packet size for the output stream.	65536	
dfs.client.block.write.retries	The packet size for the output stream.	3	
dfs.client.close.file.timeout	The timeout interval to the output stream to wait for close file operation completion.	3600000 (ms)	
dfs.client.block.write.timeout	Time of timeout interval to the output stream to write data into socket.	3600000 (ms)	
dfs.prefetchsize	The number of blocks which metadatas will be prefetched.	10	
dfs.client_local_block_read_buffer	The buffer size if read block from local file system instead of network.	1048576	

dfs.client.socket.write.timeout	The timeout interval for socket write operation.	3600000 (ms)	
dfs.client.socket.read.timeout	The timeout interval for socket write operation.	3600000 (ms)	
dfs.client.socket.connect.timeout	The timeout interval to setup a tcp socket connection, include resolve host name.	3600000 (ms)	
dfs.client.read.verifychecksum	Input stream will verify checksum.	true	
dfs.client.enable.read.from.loca	Enable input stream to read block directly from local file system. Need to configure on server.	true	
dfs.log.file.prefix	The libhdfs3 log file prefix, see glog document.	libhdfs3	
dfs.log.stderr	Output libhdfs3 log to stderr.	true	
dfs.replication	The default number of replica.	3	
dfs.blocksize	The default block size, can be overwritten when create output stream.	67108864	