Pivotal™

Pivotal HD Enterprise 1.0
Stack and Tool Reference Guide

Rev: A03

Table of Contents

Preface

This guide provides reference information for system administrators and database superusers responsible for installing a Pivotal HD Enterprise system.

- About This Guide
- Getting Support

About Pivotal, Inc.

Greenplum is currently transitioning to a new corporate identity (Pivotal, Inc.). We estimate that this transition will be completed in 2013. During this transition, there will be some legacy instances of our former corporate identity (Greenplum) appearing in our products and documentation. If you have any questions or concerns, please do not hesitate to contact us through our web site: `http://gopivotal.com/about-pivotal/support`.

About This Guide

This guide provides information and instructions for manually installing, configuring, and using a Pivotal HD Enterprise system. This guide is intended for system and database administrators responsible for managing a Pivotal HD system.

This guide assumes knowledge of Linux/UNIX system administration, database management systems, database administration, and structured query language (SQL).

Because HAWQ, ships with Pivotal HD is based on PostgreSQL, this guide assumes some familiarity with PostgreSQL.

Getting Support

Pivotal support, product, and licensing information can be obtained as follows.

Product information and Technical Support

For technical support, documentation, release notes, software updates, or for information about Pivotal products, licensing, and services, go to `www.gopivotal.com`.

Additionally you can still obtain product and support information from the EMC Support Site at: `http://support.emc.com`.

1. MapReduce V1 (MR1) Based Clusters

This chapter is applicable for those customers who want to deploy a MR1-based cluster instead of a Yarn-based cluster. If you are deploying a Yarn-based cluster, refer to Chapter 2, “Yarn-Based Clusters”.

> Pivotal HD 1.0.1 supports YARN (MR2) resource manager by default. If you don't want to deploy a YARN based cluster, we provide MR1 as optional manually-installable software.
>
> Note that since MR1 needs to be installed manually, you won't be able to use Pivotal Command Center for monitoring and management of the cluster.

This chapter includes information and manual installation instructions (where applicable) about the following individual components:

- HDFS: HDFS 2.0.2_alpha
- MR1: MapReduce 1.0.3

Once you have completed installation and configuration of these MR1 components; you can complete your manual stack installation using the instructions provided in Chapter 3, “Additional Stack Components for Manual Installations”.

Prerequisities

- Oracle Java Development Kit (JDK) 1.6. Oracle JDK must be installed on every machine before getting started on each Hadoop component.
- You must ensure that time synchronization and DNS are functioning correctly on all client and server machines. For example, you can run the following command to sync the time with NTP server:

```
# service ntpd stop; ntpdate 10.32.97.146; service ntpd start
```

Installation Notes

In this chapter we use `yum` to install the packages, however you can also install them by running `rpm -ivh package_name-<version>.rpm`. For how to create the Yum repository, refer to Appendix A, “Creating a YUM EPEL Repository”

In this document `nn` within rpm file names represents the rpm build number. It is different for different components.

Hadoop

Pivotal HD 1.0.1 Hadoop MR1 installs 2 main components: HDFS and MR1.

Prerequisites

To install PHD 1.0.1 Hadoop, you first need to install the dependent packages on all the cluster nodes:

```
# yum install nc
# yum install bigtop-utils
# yum install zookeeper
```

Core Pacakge

Install the Hadoop Core package on all the nodes:

```
# yum install hadoop
```

HDFS

Namenode

Install HDFS core package with Namenode daemon package and dependencies on HDFS Namenode machine:

```
# yum install bigtop-jsvc
# yum install hadoop-hdfs
# yum install hadoop-hdfs-namenode
```

SecondaryNamenode

Install HDFS core package with SecondaryNamenode daemon package and dependencies on HDFS SecondaryNamenode machine:

```
# yum install bigtop-jsvc
# yum install hadoop-hdfs
# yum install hadoop-hdfs-secondarynamenode
```

Datanode

Install HDFS core package with Datanode daemon package and dependencies on all the HDFS Datanode machines:

```
# yum install bigtop-jsvc
# yum install hadoop-hdfs
# yum install hadoop-hdfs-datanode
```

MR1

JobTracker

Install MR1 core package with JobTracker daemon package on MR1 JobTracker machine:

```
# yum install hadoop-mr1
# yum install hadoop-mr1-jobtracker
```

TaskTracker

Install MR1 core package with TaskTracker daemon package on all the MR1 TaskTracker machines:

```
# yum install hadoop-mr1
# yum install hadoop-mr1-tasktracker
```

Hadoop Configuration

By default, Hadoop create a set of configuration files with empty properties at `/etc/gphd/hadoop/conf.empty`, which is linked from symbolic `/etc/gphd/hadoop/conf/`.

Pseudo Mode

If you want to run Hadoop MR1 as pseudo-distributed mode on one single host, you can go through all the above installation steps on one single host and then install the `hadoop-mr1-conf-pseud`o package:

```
# yum install hadoop-mr1-conf-pseudo
```

Distributed Mode

If you want to run Hadoop MR1 as distributed mode, you can create a new directory on `/etc/gphd/hadoop`, such as `conf.dist` and change the target of symbolic link `/etc/gphd/hadoop/conf/` to the new create directory.

```
# mkdir -p /etc/gphd/hadoop/conf.dist
# unlink /etc/gphd/hadoop/conf
# ln -sf /etc/gphd/hadoop/conf.dist /etc/gphd/hadoop/conf
# cd /etc/gphd/hadoop
# cp conf.empty/* conf.dist/
# cd conf.dist
```

COMMON configuration

First, set `fs.defaultFS` (`fs.default.file` is depressed) in the `core-site.xml`.

core-site.xml

```
<property>
   <name>fs.defaultFS</name>
   <value>hdfs://<namenode-host>:8020</value>
</property>
```

Note: Replace the *`<namenode-host>`* with the actual namenode hostname or IP.

HDFS configuration

By default, hadoop-hdfs creates a default data directory `/var/lib/gphd/hadoop-hdfs-<version>` with owner `hdfs:hadoop`. You can use it to store the local directories of namenode, secondarynamenode and datanode. Then, when the namenode, secondarynamenode or datanode daemon start, they will automatically create the right subfolders with correct permission and owner under the data directory.

If you specify the new data directories, you need to create and configure them with owner `hdfs:hadoop` and correct permissions first.

Here, take the default directory as example.

hdfs-site.xml

```
<property>
   <name>dfs.namenode.name.dir</name>

   <value>file:///var/lib/gphd/hadoop-hdfs-2.0.2_alpha_gphd_
   2_0_1_0/cache/${user.name}/dfs/name</value>
</property>
<property>
   <name>dfs.namenode.checkpoint.dir</name>

   <value>file:///var/lib/gphd/hadoop-hdfs-2.0.2_alpha_gphd_
   2_0_1_0/cache/${user.name}/dfs/namesecondary</value>
</property>
<property>
   <name>dfs.datanode.data.dir</name>
   <value>file:///var/lib/gphd/hadoop-hdfs-2.0.2_alpha_gphd_
   2_0_1_0/cache/${user.name}/dfs/data</value>
</property>
```

MR1 configuration

By default, hadoop-mr1 create default data directory `/var/lib/gphd/hadoop-mr1-<version>` with owner `mapred:hadoop`. You can configure it as local storage directories for MR1 daemons to store temporary data and intermediate map output files while running MapReduce jobs (refer to `property: mapred.local.dir`).

If you specify the new data directories, you need to create and configure them with owner `mapred:hadoop` and correct permissions first.

Here, take the default directory as example.

mapred-site.xml

```
<property>
   <name>mapred.job.tracker</name>
   <value><jobtracker-host>:8021</value>
</property>
<property>
   <name>mapred.system.dir</name>
   <value>/mapred/system</value>
</property>
<property>
   <name>mapred.local.dir</name>
   <value>/var/lib/gphd/hadoop-mr1-1.0.3_gphd_2_0_1_0/cache/
```

```
    ${user.name}/tasks</value>
</property>
<property>
    <name>mapreduce.jobtracker.staging.root.dir</name>
    <value>/user</value>
</property>
```

Note: Replace the `<jobtracker-host>` with actual jobtracker hostname or IP.

Usage

Starting/Stopping HDFS daemons

HDFS includes three main components: NameNode, DataNode, Secondary NameNode.

Before starting the daemons, make sure the user `hdfs` has write permission on related folders, which are configured in `hdfs-site.xml`.

Note: Since the `hadoop-hdfs` already created the data folder `/var/lib/gphd/hadoop-hdfs-<version>` with owner `hdfs:hadoop`, you can skip this step here. However, if you want to specify a new one, create it and change the owner to `hdfs:hadoop` first.

Starting HDFS daemons

Start the NameNode daemon:

You need to format the NameNode before starting it:

```
# sudo -u hdfs hdfs namenode -format
```

Note: You only have to do this once.

Then start the NameNode:

```
# service hadoop-hdfs-namenode start
```

When NameNode is started, you can visit its dashboard at: `http://namenode-host:50070`

Start the DataNode daemon:

```
# service hadoop-hdfs-datanode start
```

Start the Secondary Namenode daemon:

```
# service hadoop-hdfs-secondarynamenode start
```

Using HDFS

When the HDFS components are started, you can try some HDFS operations and create some folders or other operations:

```
/* you can see a full list of hdfs dfs command options */
# hdfs dfs

# sudo -u hdfs hdfs dfs -ls /
# sudo -u hdfs hdfs dfs -mkdir /tmp
# sudo -u hdfs hdfs dfs -chmod 1777 /tmp
```

Note: By default, the root folder is owned by user `hdfs`, so you have to use `sudo -u hdfs ***` to execute the admin commands.

Shutting Down HDFS

Stop the NameNode Daemon:

```
# service hadoop-hdfs-namenode stop
```

Stop the DataNode Daemon:

```
# service hadoop-hdfs-datanode stop
```

Stop the Secondary Namenode Daemon:

```
# service hadoop-hdfs-secondarynamenode stop
```

Starting/Stopping MR1 daemons

MR1 also includes two services: JobTracker (JT) and TaskTracker (TT).

Before starting the daemons, make sure the user `mapred` has write permission on related folders, which are configured in `mapred-site.xml`.

Note: Since the `hadoop-mr1` already created the data folder `/var/lib/gphd/hadoop-mr1-<version>` with owner `mapred:hadoop`, so you can skip this step here. However, if the you want to specify a new one, create it and change the owner to `mapred:hadoop` first.

Create working directories on HDFS

You need to create some directories on HDFS that are needed by `hadoop-mr1`.

```
// create mapred.system.dir
# sudo -u hdfs hdfs dfs -mkdir -p /mapred/system
# sudo -u hdfs hdfs dfs -chown -R mapred:hadoop /mapred

// create mapreduce.jobtracker.staging.root.dir staging
directory
# sudo -u hdfs hdfs dfs -mkdir -p /user
```

Verify the HDFS File Structure

```
# sudo -u hdfs hdfs dfs -ls -R /
```

You should see:

```
drwxr-xr-x   - mapred hadoop          0 2013-04-18 04:20
/mapred
drwxr-xr-x   - mapred hadoop          0 2013-04-18 04:20
/mapred/system
drwxrwxrwx   - hdfs   supergroup          0 2013-04-18 04:20
/tmp
drwxrwxrwx   - hdfs   supergroup          0 2013-04-18 04:18
```

```
/user
```

Start MR1 Daemons

Start the JobTracker Daemon

Start the JobTracker daemon only on the master node:

```
# service hadoop-mr1-jobtracker start
```

When JobTracker is started, you can visit its dashboard at:
`http://jobtracker-host:50030/`

Start the TaskTracker Daemon

Start the TaskTracker daemon on all hosts that will be used as working nodes:

```
# service hadoop-mr1-tasktracker start
```

Using MR1

After JobTracker and TaskTracker are started, you can now submit MapReduce Jobs.

Note: Make sure HDFS daemons are running, and create the home directory `/user/$<`*`user.name`*`>` for each MapReduce user on hdfs. Here we use the user `hadoop`.

```
# sudo -u hdfs hdfs dfs -mkdir -p /user/hadoop
# sudo -u hdfs hdfs dfs -chown hadoop:hadoop /user/hadoop
```

Submit an example MapReduce job to generate PI:

```
# su - hadoop
$ hadoop-mr1 jar
/usr/lib/gphd/hadoop-mr1-1.0.3_gphd_2_0_1_0_dev/hadoop-examp
les-*.jar pi 2 10000
```

This will run the PI generation example. You can track the progress of this job at the JobTracker dashboard: `http://jobtracker-host:50030/`.

Stop MR1

You can stop the MR1 daemons manually by using the following commands.

Stop the TaskTracker Daemon

```
# service hadoop-mr1-tasktracker stop
```

Stop the JobTracker Daemon

```
# service hadoop-mr1-jobtracker stop
```

Now you have completed installation of MR1 components; you can complete your manual stack installation using the instructions provided in Chapter 3, “Additional Stack Components for Manual Installations”.

2. Yarn-Based Clusters

This chapter is applicable for those customers who are manually deploying a Yarn-based cluster. If you are deploying a MapReduce V1 (MR1)-based cluster, refer to Chapter 1, “MapReduce V1 (MR1) Based Clusters”.

Pivotal HD 1.0.1 supports YARN (MR2) resource manager by default. If you don't want to deploy a YARN based cluster, we provide MR1 as optional manually-installable software.

This chapter describes how to manually install, configure, and use the Pivotal HD 1.0.1 distribution of the following Apache Hadoop components:

- HDFS
- YARN
- MapReduce

Once you have completed installation and configuration of these YARN components; you can complete your manual stack installation using the instructions provided in Chapter 3, “Additional Stack Components for Manual Installations”.

Prerequisities

- Oracle Java Development Kit (JDK) 1.6. Oracle JDK must be installed on every machine before getting started on each Hadoop component.
- You must ensure that time synchronization and DNS are functioning correctly on all client and server machines. For example, you can run the following command to sync the time with NTP server:

```
# service ntpd stop; ntpdate 10.32.97.146; service ntpd start
```

Installation Notes

In this chapter we install packages by running `rpm -ivh <package_name.-<version>.rpm.` You can also install them by using `yum install <package_name>` if you already have yum repository setup (see Appendix A, “Creating a YUM EPEL Repository”).

In this document `nn` within rpm file names represents the rpm build number. It is different for different components.

Hadoop

This section provides instructions for installing each of the following core Hadoop RPMs:

- HDFS Namenode Setup
- HDFS Datanode Setup

- HDFS Secondary Namenode Setup
- YARN
- YARN NodeManager Setup
- Mapreduce HistoryServer Setup

Hadoop RPM Packages

Pivotal provides the following RPMs as part of this release. The core packages provide all executables, libraries, configurations, and documentation for Hadoop and is required on every node in the Hadoop cluster as well as the client workstation that will access the Hadoop service. The daemon packages provide a convenient way to manage Hadoop daemons as Linux services, which rely on the core package.

hadoop-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Core
Requires	bigtop-utils, zookeeper-core
Description	Hadoop core packages provides the common core packages for running Hadoop
Install on Nodes	Every node in the Hadoop cluster and the client workstation that will access the Hadoop service.

hadoop-hdfs-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Core
Requires	hadoop, bigtop-jsvc
Description	Hadoop HDFS core packages provides the common files for running HFS.
Install on Nodes	Every node in the HDFS cluster and the client workstation that will access the HDFS.

hadoop-hdfs-namenode-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-hdfs
Description	Daemon scripts package for Hadoop Namenode, which provides a convenient method to manage Namenode start/stop as a Linux service.
Install on Nodes	Only on HDFS Namenode server.

hadoop-hdfs-datanode-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-hdfs
Description	Daemon scripts package for Hadoop Datanode, which provides a convenient method to manage datanode start/stop as a Linux service.
Install on Nodes	Install on all HDFS Datanodes.

hadoop-hdfs-secondarynamenode-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-hdfs
Description	Daemon scripts package for Hadoop SecondaryNamenode, which provides a convenient method to manage SecondaryNamenode start/stop as a Linux service.
Install on Nodes	Install on one server that will be acting as the Secondary Namenode.

hadoop-hdfs-journalnode-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-hdfs
Description	Daemon scripts package for Hadoop JournalNode, which provides a convenient method to manage journalnode start/stop as a Linux service.
Install on Nodes	Install on all HDFS JournalNodes.

hadoop-hdfs-zkfc-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-hdfs
Description	Daemon scripts package for Hadoop zkfc, which provides a convenient method to manage zkfc start/stop as a Linux service.
Install on Nodes	Install on all HDFS zkfc nodes.

hadoop-hdfs-fuse-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Core
Requires	hadoop-libhdfs, hadoop-client

hadoop-hdfs-fuse-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Description	Binaries that can be used to mount hdfs as a local directory.
Install on Nodes	Install on the servers that want to mount the HDFS.

hadoop-libhdfs-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Core
Requires	
Description	Native implementation of the HDFS.
Install on Nodes	Install on servers that you want to run native hdfs.

hadoop-httpfs-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Core
Requires	bigtop-tomcat, hadoop, hadoop-hdfs
Description	HttpFS is a server that provides a REST HTTP gateway supporting all HDFS File System operations (read and write).
Install on Nodes	Install on servers that will be serving REST HDFS service

hadoop-yarn-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Core
Requires	hadoop
Description	Hadoop YARN core packages provides common files for running YARN.
Install on Nodes	Install on all YARN nodes.

hadoop-yarn-resourcemanager-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-yarn
Description	Daemon scripts package for Hadoop YARN ResourceManager, which provides a convenient method to manage ResourceManager start/stop as a Linux service.
Install on Nodes	Install on the Resource Manager node.

hadoop-yarn-nodemanager-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-yarn
Description	Daemon scripts package for Hadoop YARN NodeManager, which provides a convenient method to manage NodeManager start/stop as a Linux service.
Install on Nodes	Install on all the Node Manager nodes.

hadoop-yarn-proxyserver-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-yarn
Description	Daemon scripts package for Hadoop YARN ProxyServer, which provides a convenient method to manage ProxyServer start/stop as a Linux service.
Install on Nodes	Install on the node that will act as a proxy server from the user to applicationmaster

hadoop-mapreduce-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Core
Requires	hadoop-yarn
Description	Hadoop Mapreduce core libraries.
Install on Nodes	Install on all ResourceManager and NodeManager nodes.

hadoop-mapreduce-historyserver-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop, hadoop-mapreduce
Description	Daemon scripts package for Hadoop MapReduce HistoryServer, which provides a convenient method to manage MapReduce HistoryServer start/stop as a Linux service.
Install on Nodes	Install on the host that will be acting as the MapReduce History Server.

hadoop-doc-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Doc
Requires	N/A

hadoop-doc-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Description	Document package provides the Hadoop document.
Install on Nodes	Install on whichever host that user want to read hadoop documentation.

hadoop-conf-pseudo-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Configuration
Requires	hadoop-hdfs-datanode, hadoop-hdfs-secondarynamenode, hadoop-yarn-resourcemanager, hadoop-hdfs-namenode, hadoop-yarn-nodemanager, hadoop-mapreduce-historyserver
Description	A set of configuration files for running Hadoop in pseudo-distributed mode on one single server.
Install on Nodes	Install on the pseudo--distributed host.

hadoop-client-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm	
Type	Library
Requires	N/A
Description	A set of symbolic link which gathers the libraries for programming Hadoop and submit Hadoop jobs.
Install on Nodes	Clients nodes that will be used to submit hadoop jobs.

Prerequisites: Core Package Setup

You must perform the following steps on all the nodes in the Hadoop cluster and its client nodes:

```
$ sudo rpm -ivh
working_dir/utility/rpm/bigtop-utils-0.4.0_gphd_2_0_1_0-nn.noarch.rpm

$ sudo rpm -ivh
working_dir/zookeeper/rpm/zookeeper-3.4.5_gphd_2_0_1_0-nn.noarch.rpm

$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm
```

Where `working_dir` is the directory where you want the rpms expanded.

HDFS Namenode Setup

Install the Hadoop Namenode package on the workstation that will serve as HDFS Namenode:

```
$ sudo rpm -ivh
working_dir/utility/rpm/bigtop-jsvc-1.0.15_gphd_2_0_1_0-nn.x86_64.rpm

$ sudo rpm -ivh
```

```
working_dir/hadoop/rpm/hadoop-hdfs-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm
```

```
$ sudo rpm -ivh
working_dir/hadooop/rpm/hadoop-hdfs-namenode-2.0.2_alpha_gphd_2_0_1_0-nn.x8
6_64.rpm
```

HDFS Datanode Setup

Install the Hadoop Datanode package on the workstation that will serve as HDFS Datanode:

```
$ sudo rpm -ivh
working_dir/utility/rpm/bigtop-jsvc-1.0.15_gphd_2_0_1_0-nn.x86_64.rpm
```

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-hdfs-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm
```

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-hdfs-datanode-2.0.2_alpha_gphd_2_0_1_0-nn.x86
_64.rpm
```

HDFS Secondary Namenode Setup

Install the Hadoop Secondary Namenode package on the workstation that will serve as HDFS Secondary Namenode:

```
$ sudo rpm -ivh
working_dir/utility/rpm/bigtop-jsvc-1.0.10_gphd_2_0_1_0-nn.x86_64.rpm
```

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-hdfs-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm
```

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-secondarynamenode-2.0.2_gphd_2_0_1_0-nn.x86_6
4.rpm
```

YARN

ResourceManager Setup

Install the YARN ResourceManager package on the workstation that will serve as YARN ResourceManager:

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-yarn-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm
```

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-yarn-resourcemanager-2.0.2_alpha_gphd_2_0_1_0
-nn.x86_64.rpm
```

YARN NodeManager Setup

Install the YARN NodeManager package on the workstation that will serve as YARN nodes:

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-yarn-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm
```

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-yarn-nodemanager-2.0.2_alpha_gphd_2_0_1_0-nn.
x86_64.rpm
```

Mapreduce HistoryServer Setup

Install the YARN Mapreduce History Manager package and its dependency packages on the workstation that will serve as MapReduce History Server:

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-yarn-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.rpm
```

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-mapreduce-2.0.2_alpha_gphd_2_0_1_0-nn.x86_64.
rpm
```

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-mapreduce-historyserver-2.0.2_alpha_gphd_2_0_
1_0-nn.x86_64.rpm
```

Hadoop Configuration

You can find the configuration files for Hadoop in the following location:

```
/etc/gphd/hadoop/conf/
```

Out of the box by default it is a symbolic link to `/etc/gphd/hadoop-version/conf.empty` template.

You can make modifications to these configuration templates or create your own configuration set. If you want to use a different configuration folder, adjust the `/etc/gphd/hadoop/conf` symbolic link to point to the folder you want to utilize at runtime.

If you want to run Hadoop 2.0 in one host in pseudo-distributed mode on one single host, you can go through all the above setup steps on your host and then install the hadoop-conf-pseudo package, as follows:

```
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-conf-pseudo-2.0.2_alpha_gphd_2_0_1_0
-nn.x86_64.rpm
```

Please refer to Apache Hadoop 2.0.2-alpha documentation for how to configure Hadoop in distributed mode. In this document, we provide instructions to start using Hadoop in pseudo-distributed mode.

Hadoop Usage

After installing the daemon package for Hadoop, you can start the daemons.

Starting HDFS

HDFS includes three main components: Namenode, Datanode, Secondary Namenode.

To start the Namenode daemon:

You need to format the Namenode before starting it, as follows:

```
$ sudo -u hdfs hdfs namenode -format
```

Note: You only have to do this once. But if you have changed the hadoop namenode configuration, you may need to run this again.

Then start the Namenode by running either:

```
$ sudo service hadoop-hdfs-namenode start
```

or:

```
$ sudo hdfs hadoop-hdfs-namenode start
```

When Namenode is started, you can visit its dashboard at: `http://localhost:50070/`

To start the Datanode daemon:

Run either:

```
$ sudo service hadoop-hdfs-datanode start
```

or:

```
$ sudo hdfs hadoop-hdfs-datanode start
```

To start the Secondary Namenode daemon:

Run either:

```
$ sudo service hadoop-hdfs-secondarynamenode start
```

or:

```
$ sudo hdfs hadoop-hdfs-secondarynamenode start
```

Using HDFS

When the HDFS components are started, you can try some HDFS usage, for example:

```
$ sudo -u hdfs hdfs dfs -ls /
$ sudo -u hdfs hdfs dfs -mkdir -p /user/hadoop
$ sudo -u hdfs hdfs dfs -chown -R hadoop:hadoop /user/hadoop
#you can see a full list of hdfs dfs command options
$ hdfs dfs
$ bin/hdfs dfs -copyFromLocal /etc/passwd /user/hadoop/
```

Note: by default, the root folder is owned by user `hdfs`, so you have to use `sudo -u hdfs ***` to execute the first few commands.

Shutting down HDFS

Stop the Namenode Daemon:

Run either:

```
$ sudo service hadoop-hdfs-namenode stop
```

or:

```
$ sudo hdfs hadoop-hdfs-namenode stop
```

Stop the Datanode Daemon:

Run either:

```
$ sudo service hadoop-hdfs-datanode stop
```

or:

```
$ sudo hdfs hadoop-hdfs-datanode stop
```

Stop the Secondary Namenode Daemon:

Run either:

```
$ sudo service hadoop-hdfs-secondarynamenode stop
```

or:

```
$ sudo hdfs hadoop-hdfs-secondarynamenode stop
```

Starting YARN

YARN includes three services: ResourceManager (RM), NodeManager (NM), MapReduce HistoryManager (MRHM). RM and NM are required, MRHM is optional. Before you start these services, you need to create some working directories that are needed by `hadoop-yarn` and `hadoop-mapreduce`, as follows:

Create working directories on HDFS

```
$ sudo -u hdfs hdfs dfs -mkdir /var/log/hadoop-yarn
$ sudo -u hdfs hdfs dfs -chown yarn:hadoop
/var/log/hadoop-yarn
$ sudo -u hdfs hdfs dfs -mkdir /tmp
$ sudo -u hdfs hdfs dfs -chmod +w /tmp
$ sudo -u hdfs hdfs dfs -mkdir /user/hadoop
$ sudo -u hdfs hdfs dfs -chown hadoop:hadoop /user/hadoop
```

Starting ResourceManager

RM daemon need to be started only on the master node.

Run either:

```
$ sudo service hadoop-yarn-resourcemanager start
```

or:

```
$ sudo yarn hadoop-yarn-resourcemanager start
```

When RM is started, you can visit its dashboard at: `http://localhost:8088/`

Starting NodeManager

NM daemon needs to be started on all hosts that will be used as working nodes.

Run either:

```
$ sudo service hadoop-yarn-nodemanager start
```

or:

```
$ sudo yarn hadoop-yarn-nodemanager start
```

Start MapReduce HistoryServer

MapReduce HistoryServer only needs to be run on the server that is meant to be the history server. It is an optional service and should only be enabled if you want to keep track of the MapReduce jobs that have been run.

Run:

```
$ sudo service hadoop-mapreduce-historyserver start
```

When the MR HistoryServer is started, you can visit its dashboard at: `http://localhost:19888/`

Using YARN

After RM and NM are started, you can now submit YARN applications.

For simplicity, we assume you are running Hadoop in pseudo-distributed mode using the default pseudo configuration.

Here we will submit an example MapReduce job:

```
$ hadoop jar
/usr/lib/gphd/hadoop-mapreduce/hadoop-mapreduce-examples-2.0
.2-alpha-gphd-2.0.1.0.jar pi 2 200
```

This will run the PI generation example. You can track the progress of this job at the RM dashboard: `http://localhost:8088/`

You can also run other MapReduce examples, the following command will print a list of available examples:

```
$ hadoop jar
/usr/lib/gphd/hadoop-mapreduce/hadoop-mapreduce-examples-2.0
.2-alpha-gphd-2.0.1.0.jar
```

Stopping YARN

You can stop the YARN daemons manually by using the following commands.

To stop the MapReduce HistoryServer Daemon:

Run:

```
$ sudo service hadoop-mapreduce-historyserver stop
```

To stop the NodeManager Daemon:

Run:

```
$ sudo service hadoop-yarn-nodemanager stop
```

To stop the ResourceManager Daemon:

Run:

```
$ sudo service hadoop-yarn-resourcemanager stop
```

Now you have completed installation of MR2 components; you can complete your manual stack installation using the instructions provided in Chapter 3, “Additional Stack Components for Manual Installations”.

3. Additional Stack Components for Manual Installations

If you are deploying an MR1-based cluster, you should have already manually installed HDFS and MR1, as described in Chapter 1, “MapReduce V1 (MR1) Based Clusters”.

If you are manually deploying a YARN-based cluster, you should have already have manually installed MR2, as described in Chapter 2, “Yarn-Based Clusters”.

This chapter describes how to manually install, configure and use other components of the Pivotal HD 1.0.1 distribution of Apache Hadoop.

- Zookeeper: Zookeeper 3.4.5
- HBase: HBase 0.94.2
- Hive: Hive 0.9.1
- Pig: Pig 0.10.1
- Mahout: Mahout 0.7
- Flume: Flume 1.3.1
- Sqoop: Sqoop 1.4.2
- Spring Data
- HVE (Hadoop Virtualization Extensions)
- Vaidya
- DataLoader

For information about the installation, configuration, and use of the USS component is provided in Chapter 4, “Unified Storage System (USS)”

Prerequisities

If not already met:

- Oracle Java Development Kit (JDK) 1.6. Oracle JDK must be installed on every machine before getting started on each Hadoop component.
- You must ensure that time synchronization and DNS are functioning correctly on all client and server machines. For example, you can run the following command to sync the time with NTP server:

```
# service ntpd stop; ntpdate 10.32.97.146; service ntpd start
```

In this document `nn` within rpm file names represents the rpm build number. It is different for different components.

Installation Notes

In this chapter we install packages by running `rpm -ivh package_name-<version>.rpm`. You can also install them by using `yum install <package_name>` if you already have yum repository setup (see Appendix A, "Creating a YUM EPEL Repository").

In this document `nn` within rpm file names represents the rpm build number. It is different for different components.

Zookeeper

This section describes how to install, configure, and use Zookeeper.

Zookeeper RPM Packages

Pivotal HD provides the following RPMs as part of this release. The core package provides all executable, libraries, configurations, and documentation for Zookeeper and is required on every node in the Zookeeper cluster as well as the client workstation that will access the Zookeeper service. The daemon packages provide a convenient way to manage Zookeeper daemons as Linux services, which rely on the core package.

Note: Zookeeper doesn't require Hadoop Core Packages.

zookeeper-3.4.5_gphd_2_0_1_0-nn.noarch.rpm	
Type	Core
Requires	N/A
Description	Zookeeper core package which provides the executable, libraries, configuration files and documentations.
Install on Nodes	Every node in the ZooKeeper cluster, and the client workstations which will access the ZooKeeper service.

zookeeper-server-3.4.5_gphd_2_0_1_0-nn.noarch.rpm	
Type	Deamon
Requires	ZooKeeper Core Package
Description	Daemon scripts package for Zookeeper server, which provides a convenient method to manage Zookeeper server start/stop as a Linux service.
Install on Nodes	N/A

zookeeper-doc-3.4.5_gphd_2_0_1_0-nn.noarch.rpm	
Type	Documentation
Requires	N/A
Description	Zookeeper documentation.
Install on Nodes	N/A

Zookeeper Server Setup

Install the Zookeeper core package and the Zookeeper server daemon package on the workstation that will serve as the zookeeper server, as follows:

```
$ sudo rpm -ivh
working_dir/zookeeper/rpm/zookeeper-3.4.5_gphd_2_0_1_0-nn.noarch.r
pm
```

```
$ sudo rpm -ivh
working_dir/zookeeper/rpm/zookeeper-server-3.4.5_gphd_2_0_1_0-nn.n
oarch.rpm
```

Where `working_dir` is the directory where you want the rpms expanded.

Zookeeper Client Setup

Install the Zookeeper core package on the client workstation to access the Zookeeper service, as follows:

```
$ sudo rpm -ivh
working_dir/zookeeper/rpm/zookeeper-3.4.5_gphd_2_0_1_0-nn.no
arch.rpm
```

Zookeeper Configuration

Zookeeper configuration files are in the following location:

```
/etc/gphd/zookeeper/conf
```

This is the default configuration for quick reference and modification. It is a symbolic link to `/etc/gphd/zookeeper-version/conf.dist` template set.

You can make modifications to these configuration templates or create your own configuration set. If you want to use a different configuration folders, adjust the symbolic link `/etc/gphd/zookeeper` to point to the folder you want to utilize at runtime.

Usage

Starting the Zookeeper Daemon

After installing the daemon package for Zookeeper, the Zookeeper server daemon will start automatically at system startup by default.

You can start the daemons manually by using the following commands.

Run:

```
$ sudo service zookeeper-server start
```

Accessing the Zookeeper service

To access the Zookeeper service on a client machine, use the command `zookeeper-client` directly in shell:

```
$ zookeeper-client
```

In the ZK shell:

```
> ls
> create /zk_test my_data
> get /zk_test
> quit
```

You can get a list of available commands by inputting "`?`" in the zookeeper shell.

Stopping the Zookeeper Daemon

You can stop the Zookeeper server daemon manually using the following commands:

Run:

```
$ sudo service zookeeper-server stop
```

HBase

This section specifies how to install, configure, and use HBase.

Prerequisites

As HBase is built on top of Hadoop and Zookeeper, the Hadoop and Zookeeper core packages must be installed for HBase to operate correctly.

HBase RPM Packages

Pivotal HD provides the following RPMs as part of this release. The core package provides all executables, libraries, configurations and documentation for HBase and is required on every node in HBase cluster as well as the client workstation that wants to access the HBase service. The daemon packages provide a convenient way to manage HBase daemons as Linux services, which rely on the core package.

hbase-0.94.2_gphd_2_0_1_0-nn.noarch.rpm	
Type	Core
Requires	Hadoop HDFS Packages and ZooKeeper Core Package
Description	HBase core package provides all executables, libraries, configuration files and documentations.

hbase-master-0.94.2_gphd_2_0_1_0-nn.noarch.rpm	
Type	Daemon
Requires	HBase Core Package
Description	Daemon scripts package for HMaster, which provides a convenient method to manage HBase HMaster server start/stop as a Linux service.

hbase-regionserver-0.94.2_gphd_2_0_1_0-nn.noarch.rpm	
Type	Daemon
Requires	HBase Core Package
Description	Daemon scripts package for HRegionServer, which provides a convenient method to manage HBase HRegionServer start/stop as a Linux service.

hbase-thrift-0.94.2_gphd_2_0_1_0-nn.noarch.rpm	
Type	Daemon (thrift service)
Requires	HBase Core Package
Description	Daemon scripts package to provide HBase service through thrift.

hbase-rest-0.94.2_gphd_2_0_1_0-nn.noarch.rpm	
Type	Daemon (Restful service)
Requires	HBase Core Package
Description	Daemon scripts package to provide HBase service through REST.

hbase-doc-0.94.2_gphd_2_0_1_0-nn.noarch.rpm	
Type	Documentation
Requires	HBase Core Package
Description	HBase documentation.

HBase Master Setup

Install the HBase core package and the HBase master daemon package on the workstation that will serve as the HMaster:

```
$ sudo rpm -ivh
```

```
working_dir/hbase/rpm/hbase-0.94.2_gphd_2_0_1_0-nn.noarch.rp
m
$ sudo rpm -ivh
working_dir/hbase/rpm/hbase-master-0.94.2_gphd_2_0_1_0-nn.no
arch.rpm
```

HBase RegionServer Setup

Install the HBase core package and the HBase regionserver daemon package on the workstation that will serve as the HRegionServer:

```
$ sudo rpm -ivh
working_dir/hbase/rpm/hbase-0.94.2_gphd_2_0_1_0-nn.noarch.rp
m
$ sudo rpm -ivh
working_dir/hbase/rpm/hbase-regionserver-0.94.2_gphd_2_0_1_0
-nn.noarch.rpm
```

HBase Client Setup

Install the HBase core package on the client workstation that will access the HBase service:

```
$ sudo rpm -ivh
working_dir/hbase/rpm/hbase-0.94.2_gphd_2_0_1_0-nn.noarch.rp
m
```

HBase Thrift Server Setup [OPTIONAL]

Install the HBase core package and the HBase thrift daemon package to provide HBase service through Apache Thrift:

```
$ sudo rpm -ivh
working_dir/hbase/rpm/hbase-0.94.2_gphd_2_0_1_0-nn.noarch.rp
m
$ sudo rpm -ivh
working_dir/hbase/rpm/hbase-thrift-0.94.2_gphd_2_0_1_0-nn.no
arch.rpm
```

REST Server Setup [OPTIONAL]

Install the HBase core package and the HBase rest daemon package to provide HBase service through Restful interface:

```
$ sudo rpm -ivh
working_dir/hbase/rpm/hbase-0.94.2_gphd_2_0_1_0-nn.noarch.rp
m
$ sudo rpm -ivh
working_dir/hbase/rpm/hbase-rest-0.94.2_gphd_2_0_1_0-nn.noar
ch.rpm
```

HBase Configuration

You can find the configuration files for HBase in the following location:

```
/etc/gphd/hbase/conf/
```

This is the default configuration for quick reference and modification.

`/etc/gphd/hbase` is a symbolic link to `/etc/gphd/`*`hbase-version`*`/`; and the conf folder is a symbolic link to the exact configuration directory.

You can make modifications to these configuration templates or create your own configuration set. If you want to use a different configuration folders, adjust the symbolic link `/etc/gphd/hbase/conf` to point to the folder you want to utilize at runtime.

HBase Post-installation Configuration

1. Login to one of the cluster nodes.
2. Create the `hbase.`*`rootdir`*
   ```
   $ sudo -u hdfs hdfs dfs -mkdir -p /hbase
   ```
3. Set permissions for the `hbase.`*`rootdir`*
   ```
   # sudo -u hdfs hdfs dfs -chown hbase:hadoop /hbase
   ```
4. Set the ownership for the `hbase.`*`rootdir`*
   ```
   $ sudo -u hdfs hadoop fs -chown hbase:hadoop /hbase
   ```
5. Add `hbase` user to the `hadoop` group if not already present using
   ```
   $ usermod -G hadoop hbase
   ```

Usage

Starting the HBase Daemon

After installing the daemon package for HBase, the HBase server daemons will start automatically at system startup by default.

You can start the daemons manually by using the following commands:

Run:

```
$ sudo service hbase-master start
```

Starting the HRegionServer daemon

You can start the daemons manually by using the following commands:

Run:

```
$ sudo service hbase-regionserver start
```

Starting the Hbase Thrift server daemon [OPTIONAL]

You can start the daemons manually by using the following commands:

Run:

```
$ sudo service hbase-thrift start
```

Starting the Hbase Rest server daemon [OPTIONAL]

You can start the daemons manually by using the following commands:

Run:

```
$ sudo service hbase-rest start
```

Accessing the HBase service

To access the HBase service on a client machine, use the command hbase directly in shell:

```
$ hbase
```

Or you can use this command to enter the hbase console:

```
$ hbase shell
```

In the HBase shell, you can run some test commands, for example:

```
hbase(main):003:0> create 'test', 'cf'
hbase(main):003:0> list 'test'
hbase(main):004:0> put 'test', 'row1', 'cf:a', 'value1'
hbase(main):005:0> put 'test', 'row2', 'cf:b', 'value2'
hbase(main):006:0> put 'test', 'row3', 'cf:c', 'value3'
hbase(main):007:0> scan 'test'
hbase(main):008:0> get 'test', 'row1'
hbase(main):012:0> disable 'test'
hbase(main):013:0> drop 'test'
hbase(main):014:0> quit
```

Type `help` to get help for the HBase shell.

Stopping the HBase daemon

You can stop the daemons manually by using the following commands:

Run:

```
$ sudo service hbase-master stop
```

Stopping the HRegionServer daemon

You can stop the daemons manually by using the following commands:

Run:

```
$ sudo service hbase-regionserver stop
```

Stopping the Hbase Thrift server daemon [OPTIONAL]

You can stop the daemons manually by using the following commands:

Run:

```
$ sudo service hbase-thrift stop
```

Stopping the Hbase Rest server daemon [OPTIONAL]

You can stop the daemons manually by using the following commands:

Run:

```
$ sudo service hbase-rest stop
```

Hive

Hive Components

A Hive installation consists of the following components:

- `hive-server`
- `hive-metastore`
- `hive-dbserver`

Prerequisites

As Hive is built on top of Hadoop, HBase and Zookeeper, the Hadoop, HBase and Zookeeper core packages must be installed for Hive to operate correctly.

The following prerequisites must be also met before installing Hive:

- PostgresSQL Server
- Hive Metastore backed by a DB Server.

  ```
  ve/gphd/warehouse
  hive.metastore.local = false
  ```

Hive RPM Packages

Hive consists of one core package and a thrift sever daemon package that provides Hive service through thrift.

hive-0.9.1_gphd_2_0_1_0-nn.noarch.rpm	
Type	Core
Requires	Hadoop, HBase Core Packages
Description	Hive core package provides the executables, libraries, configuration files and documentations.
Install on Nodes	Hive Client workstation

hive-server-0.9.1_gphd_2_0_1_0-nn.noarch.rpm	
Type	Daemon (thrift server)
Requires	Hive Core Package
Description	Daemon scripts package to provide Hive service through thrift
Install on Nodes	Hive Thrift server node

hive-metastore-0.9.1_gphd_2_0_1_0-nn.noarch.rpm	
Type	Deamon (Metastore server)
Requires	Hive Core Package
Description	Daemon scripts package to provide Hive metadata information through metastore server.
Install on Nodes	Hive Metastore server node

Installing Hive

Set up PostgreSQL on the HIVE_METASTORE Node

1. Choose one of the cluster nodes to be the `HIVE_METASTORE`.
2. Login to the nominated `HIVE_METASTORE` node as root.
3. Execute the following commands
   ```
   yum install postgresql-server
   ```
4. Initialize the database:
   ```
   service postgresql initdb
   ```
5. Open the `/var/lib/pgsql/data/postgresql.conf` file and set the following values:
   ```
   listen_addresses = '*'
   standard_conforming_strings = off
   ```
6. Open the `/var/lib/pgsql/data/pg_hba.conf` file and comment out all the lines starting with `host` and `local` by adding `#` to start of the line.
   Add following lines:
   ```
   local   all        all                                      trust
   host    all        all         0.0.0.0  0.0.0.0              trust
   ```
7. Create `/etc/sysconfig/pgsql/postgresql` file and add
   ```
   PGPORT=10432
   ```
8. Start the database:
   ```
   service postgresql start
   ```
9. Create the user, database.
   ```
   sudo -u postgres createuser -U postgres -p 10432 -a hive
   sudo -u postgres createdb -U postgres -p 10432 metastore
   ```

Set up the HIVE_METASTORE

1. Install Hive metastore using:
   ```
   yum install hive-metastore postgresql-jdbc
   ```

2. Open the `/etc/gphd/hive/conf/hive-site.xml` and change it to following:

```
<configuration>
<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hive</value>
</property>
<property>
    <name>hive.metastore.uris</name>
    <value>thrift://<CHANGE_TO_
HIVE_METASTORE_ADDRESS>:9083</value>
</property>
<property>
    <name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:postgresql://<CHANGE_TO_HIVE_METASTORE_ADDRESS>:
10432/metastore</value>
</property>
<property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/hive/gphd/warehouse</value>
</property>
<property>
    <name>hive.hwi.war.file</name>

<value>/usr/lib/gphd/hive/lib/hive-hwi-0.9.1-gphd-2.0.1.0.wa
r</value>
</property>
<property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>org.postgresql.Driver</value>
</property>
<property>
    <name>datanucleus.autoCreateSchema</name>
    <value>false</value>
</property>
<property>
    <name>hive.metastore.local</name>
    <value>false</value>
</property>
<property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
</property>
```

```
<property>
    <name>hive.metastore.execute.setugi</name>
    <value>true</value>
</property>
</configuration>
```

Note: Replace *<CHANGE_TO_HIVE_METASTORE_ADDRESS>* in above file.

3. Create file `/etc/gphd/hive/conf/hive-env.sh` and add the following:

   ```
   export HADOOP_HOME="/usr/lib/gphd/hadoop"
   export HADOOP_CONF_DIR="/etc/gphd/hadoop/conf"
   export HADOOP_MAPRED_HOME="/usr/lib/gphd/hadoop-mapreduce"
   export HIVE_CONF_DIR="/etc/gphd/hive/conf"
   ```

   Make it executable using:

   ```
   chmod +x /etc/gphd/hive/conf/hive-env.sh
   ```

4. Edit file `/etc/gphd/hadoop/conf/hadoop-env.sh` and add the following before `export HADOOP_CLASSPATH`:

   ```
   export HIVELIB_HOME=$GPHD_HOME/hive/lib
   export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:\
   $HIVELIB_HOME/hive-service-0.9.1-gphd-2.0.1.0.jar:\
   $HIVELIB_HOME/libthrift-0.7.0.jar:\
   $HIVELIB_HOME/hive-metastore-0.9.1-gphd-2.0.1.0.jar:\
   $HIVELIB_HOME/libfb303-0.7.0.jar:\
   $HIVELIB_HOME/hive-common-0.9.1-gphd-2.0.1.0.jar:\
   $HIVELIB_HOME/hive-exec-0.9.1-gphd-2.0.1.0.jar:\
   $HIVELIB_HOME/postgresql-jdbc.jar
   ```

5. Link postgresql jar:

   ```
   ln -s /usr/share/java/postgresql-jdbc.jar
   /usr/lib/gphd/hive/lib/postgresql-jdbc.jar
   ```

6. Create the schema:

   ```
   sudo -u postgres psql -U hive -d metastore -p 10432 -f
   /usr/lib/gphd/hive-0.9.1_gphd_2_0_1_0/scripts/metastore/upgr
   ade/postgres/hive-schema-0.9.0.postgres.sql
   ```

7. Start the hive-metastore:

   ```
   service hive-metastore start
   ```

Note: MySQL is no longer supported. Please migrate from MySQL to PostgreSQL.

Hive Client Setup

Hive is a Hadoop client-side library. Install the Hive core package on the client workstation:

```
$ sudo yum install hive
```

Hive Thrift Server Setup [OPTIONAL]

Install the Hive core package and Hive thrift daemon package to provide Hive service through thrift.

```
$ sudo yum install hive-server
```

All the steps to start server are same. Replace hive-metastore with hive-server. There is no need to install postgres

Hive MetaStore Server Setup [OPTIONAL]

Install the Hive core package and Hive Metastore daemon package to provide Hive metadata information through centralized Metastore service:

```
$ sudo yum install hive-metastore
```

Hive Configuration

You can find the configuration files for Hive in the following location:

```
/etc/gphd/hive/conf/
```

This is the default configuration for quick reference and modification. It is a symbolic link to `/etc/gphd/hive-version/conf`

You can make modifications to this configuration template or create your own. If you want to use a different configuration folder, adjust the symbolic link `/etc/gphd/hive/conf` to point to the folder you want to utilize at runtime.

Hive Post-installation Configuration

1. Login to one of the cluster nodes as root.
2. Create the `hive.warehouse.dir`
   ```
   $ sudo -u hdfs hadoop fs -mkdir -p /hive/gphd/warehouse
   ```
3. Set permissions for the `hive.warehouse.dir`
   ```
   $ sudo -u hdfs hadoop fs -chmod 775 /hive/gphd/warehouse
   ```
4. Set the ownership for the `hive.warehouse.dir`
   ```
   $ sudo -u hdfs hadoop fs -chown hadoop:hadoop
   /hive/gphd/warehouse
   ```
5. Add hive user to hadoop group if not already present using
   ```
   $ usermod -G hadoop hive
   ```

Hive Usage

Start Hive Client

To run Hive on a client machine, use the hive command directly in shell:

```
$ hive
```

You can check the Hive command usage by:

```
$ hive -help
```

Start/Stop Hive Thrift Server [Optional]

You can start/stop Hive thrift server daemon as follows:

Run:

```
$ sudo service hive-server start
$ sudo service hive-server stop
```

Start/Stop Hive Metastore Server [Optional]

You can start/stop Hive Metastore server daemon as follows:

Run:

```
$ sudo service hive-metastore start
$ sudo service hive-metastore stop
```

Configuring a Secure Hive Cluster

If you are running Hive in a standalone mode using a local or embedded MetaStore you do not need to make any modifications.

The Hive MetaStore supports Kerberos authentication for Thrift clients. Follow the instructions provided in the section about "Security" on page 82 to configure Hive for a security-enabled HD cluster.

Pig

This section specifies how to install, configure, and use Pig.

Prerequisites

As Pig is built on top of Hadoop the Hadoop package must be installed to run Pig correctly.

Pig RPM Packages

Pig has only one core package.

pig-0.10.1_gphd_2_0_1_0-nn.noarch.rpm	
Type	Core
Requires	Hadoop Core Packages
Description	Pig core package provides executable, libraries, configuration files and documentation.
Install on Nodes	Pig client workstation

pig-doc-0.10.1_gphd_2_0_1_0-nn.noarch.rpm	
Type	Documentation
Requires	N/A
Description	Pig documentation.
Install on Nodes	N/A

Pig Client Setup

Pig is a Hadoop client-side library. Install the Pig package on the client workstation:

```
$ sudo rpm -ivh
working_dir/pig/rpm/pig-0.10.1_gphd_2_0_1_0-nn.noarch.rpm
```

Pig Configuration

You can find the configuration files for Pig in the following location:

```
/etc/gphd/pig/conf/
```

This is the default configuration templates for quick reference and modification.

You can modify these configuration templates or create your own configuration set. If you want to use a different configuration folder, adjust the symbolic link conf under `/etc/gphd/pig/` to point to the folder you want to utilize at runtime.

Usage

To run Pig scripts on a client machine, use the command pig directly in shell:

```
$ pig COMMAND
```

You can check the pig command usage by:

```
$ pig -help
```

Mahout

This section specifies how to install, configure, and use Mahout.

Prerequisites

Mahout is built on top of Hadoop, so the Hadoop package must be installed to get Mahout running.

Mahout RPM Packages

Mahout has only one core package.

mahout-0.7_gphd_2_0_1_0-nn.noarch.rpm	
Type	Core
Requires	Hadoop Core Packages
Description	Mahout core package provides executable, libraries, configuration files and documentations.
Install on Nodes	Mahout clie.nt workstation

Pig Client Setup

Mahout is a Hadoop client-side library. Install the Mahout package on the client workstation:

```
$ sudo rpm -ivh
working_dir/mahout/rpm/mahout-0.7_gphd_2_0_1_0-nn.noarch.rpm
```

Pig Configuration

You can find the configuration files for Mahout in the following location:

```
/etc/gphd/mahout/conf/
```

This is the default configuration templates for quick reference and modification.

You can modify these configuration templates or create your own configuration set. If you want to use a different configuration folder, adjust the symbolic link conf under `/etc/gphd/mahout/` to point to the folder you want to utilize at runtime.

Usage

To run Mahout scripts on a client machine, use the command mahout directly in shell:

```
$ mahout PROGRAM
```

You can check the full list of mahout programs by running:

```
$ mahout
```

Flume

Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application. For more info, please refer to the Apache Flume page: http://flume.apache.org/

This section specifies how to install, configure, and use Flume.

Prerequisites

As Flume is built on top of Hadoop, the Hadoop package must be installed to get Flume running correctly.

(Hadoop core and hadoop hdfs should be installed)

Flume RPM Packages

Flume consists of one core package and a flume-agent sever daemon package.

flume-1.3.1_gphd_2_0_1_0-nn.noarch.rpm	
Type	Core
Requires	Hadoop Core Packages
Description	Flume core package provides executable, libraries, configuration files and documentations.
Install on Nodes	Flume client workstation.

flume-agent-1.3.1_gphd_2_0_1_0-nn.noarch.rpm	
Type	Daemon (Flume Agent server)
Requires	Flume core Package
Description	Daemon scripts package to provide Flume service for generating, processing, and delivering data.
Install on Nodes	Flume agent server node.

Flume Client Setup

Flume is a Hadoop client-side library. Install the Flume package on the client workstation: Installation can be done using yum or rpm `-ivh` options.

```
$ sudo rpm -ivh
working_dir/flume/rpm/flume-1.3.1_gphd_2_0_1_0-nn.noarch.rpm
```

Note: User `flume` and group `flume` should be created with correct configuration, including `uid`, `gid`, `home_dir` and `shell`. Check in following paths: `/etc/passwd`, `/etc/group`

Flume Agent Setup [Optional]

Install the Flume core package and Flume agent daemon package to provide Flume service for generating, processing, and delivering data:

```
$ sudo rpm -ivh
working_dir/flume/rpm/flume-1.3.1_gphd_2_0_1_0-nn.noarch.rpm
$ sudo rpm -ivh
```

```
working_dir/flume/rpm/flume-agent-1.3.1_gphd_2_0_1_0-nn.noar
ch.rpm
```

Flume Configuration

You can find the configuration files for Flume in the following location:

```
/etc/gphd/flume/conf/
```

This is the default configuration templates for quick reference and modification.

You can modify these configuration templates or create your own configuration set. If you want to use a different configuration folder, adjust the symbolic link conf under `/etc/gphd/flume/` to point to the folder you want to utilize at runtime.

Usage

Starting Flume Client

To run Flume scripts on a client machine, use the command `flume-ng` directly in shell:

```
$ flume-ng
```

You can check the `flume-ng` command usage by running:

```
$ flume-ng --help
```

Starting/Stopping Flume Agent Server [Optional]

You can start/stop Flume agent server daemon as follows:

Run:

```
$ sudo service flume-agent start
$ sudo service flume-agent stop
$ sudo service flume-agent status
```

Sqoop

Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases. For more details, please refer to the Apache Sqoop page: http://sqoop.apache.org/

This section specifies how to install, configure, and use Sqoop.

Prerequisites

As Sqoop is built on top of Hadoop and HBase, the Hadoop and HBase package must be installed to get Flume running correctly.

Sqoop RPM Packages

Flume consists of one core package and a sqoop-metastore sever daemon package.

sqoop-1.4.2_gphd_2_0_1_0-nn.noarch.rpm	
Type	Core
Requires	Hadoop, HBase Core Packages
Description	Sqoop core package provides executable, libraries, configuration files and documentations.
Install on Nodes	Sqoop. client workstation

.

sqoop-metastore-1.4.2_gphd_2_0_1_0-nn.noarch.rpm	
Type	Daemon (Sqoop Metastore server)
Requires	Sqoop core Package
Description	Daemon scripts package to provide shared metadata repository for Sqoop.
Install on Nodes	Sqoop metastore server node

Sqoop lient Setup

Sqoop is a Hadoop client-side library. Install the Sqoop package on the client workstation:

```
$ sudo rpm -ivh
working_dir/sqoop/rpm/sqoop-1.4.2_gphd_2_0_1_0-nn.noarch.rpm
```

Note: User `sqoop` and group `sqoop` should be created with correct configuration: `uid sqoop`, `gid sqoop`, `homedir /home/sqoop`, `shell /sbin/nologin`. Check in following path: `/etc/passwd and /etc/group .`

Sqoop Metastore Setup [Optional]

Install the Sqoop core package and Sqoop agent daemon package to provide shared metadata repository for Sqoop. `sqoop-metastore` has the dependency with sqoop-core package:

```
$ sudo rpm -ivh
working_dir/sqoop/rpm/sqoop-1.4.2_gphd_2_0_1_0-nn.noarch.rpm
$ sudo rpm -ivh
working_dir/sqoop/rpm/sqoop-metastore-1.4.2_gphd_2_0_1_0-nn.
noarch.rpm
```

Sqoop Configuration

You can find the configuration files for Flume in the following location:

```
/etc/gphd/sqoop/conf/
```

This is the default configuration templates for quick reference and modification.

You can modify these configuration templates or create your own configuration set. If you want to use a different configuration folder, adjust the symbolic link conf under `/etc/gphd/sqoop/` to point to the folder you want to utilize at runtime.

Usage

Starting Sqoop Client

To run Sqoop scripts on a client machine, use the command sqoop directly in shell:

```
$ sqoop
```

You can check the sqoop command usage by running:

```
$ sqoop help
```

Starting/Stopping Sqoop Metastore Server [Optional]

You can start/stop Sqoop metastore server daemon as follows:

Run:

```
$ sudo service sqoop-metastore start
$ sudo service sqoop-metastore stop
$ sudo service sqoop-metastore status
```

Spring Data

Spring for Apache Hadoop provides support for writing Apache Hadoop applications that benefit from the features of Spring, Spring Batch, and Spring Integration. For more information, please refer to the Spring Data official page: http://www.springsource.org/spring-data/hadoop

Installing Spring Data Hadoop

1. Download and copy Pivotal HD Tools Tarball to `/home/gpadmin/`. Make sure the tarball has read permission for user gpadmin. To extract the PHDTools tarball execute the following command:

```
[root@hdp2-w17 gpadmin]# chown gpadmin:gpadmin
PHDTools-version.tar.gz
[root@hdp2-w17 gpadmin]# ls -lrt PHDTools-version.tar.gz
-rw-r--r-- 1 gpadmin gpadmin 499930679 Mar 20 20:12
PHDTools-version.tar.gz

[root@hdp2-w17 gpadmin]# sudo su - gpadmin

[gpadmin@hdp2-w17 ~]$ tar xzvf PHDTools-version.tar.gz
[gpadmin@hdp2-w17 ~]$ ls -lrt GPHD*
drwxrwxr-x 5 gpadmin gpadmin      4096 Mar 20 00:35
PHDTools-version
-rw-r--r-- 1 gpadmin gpadmin 499930679 Mar 20 20:12
```

```
PHDTools-version.tar.gz

[gpadmin@hdp2-w17 ~]$ cd
PHDTools-version/spring-data-hadoop/rpm/
[gpadmin@hdp2-w17 rpm]$ ls -lrt
total 1580
-rw-rw-r-- 1 gpadmin gpadmin 1610604 Mar 20 00:04
spring-data-hadoop-1.0.1.RC1-3.noarch.rpm
-rw-rw-r-- 1 gpadmin gpadmin      76 Mar 20 00:44
spring-data-hadoop-1.0.1.RC1-3.noarch.rpm.md5
```

Installing Spring Data Hadoop through RPM

To install Spring Data Hadoop through RPM execute the following command:

```
[gpadmin@hdp2-w17 rpm]$ pwd
/home/gpadmin/PHDTools-version/spring-data-hadoop/rpm

[gpadmin@hdp2-w17 rpm]$ sudo rpm -ivh
spring-data-hadoop-1.0.1.RC1-3.noarch.rpm
Preparing...
########################################### [100%]
   1:spring-data-hadoop
########################################### [100%]
[gpadmin@hdp2-w17 rpm]$ sudo rpm -qa |grep spring
spring-data-hadoop-1.0.1.RC1-3.noarch
```

Spring Data Hadoop

By default, Spring Data Hadoop is installed to `/usr/local/gphd/` directory.

The following documentation is installed:

```
[gpadmin@hdp2-w17 ~]$ cd
/usr/local/gphd/spring-data-hadoop-1.0.1.RC1
[gpadmin@hdp2-w17 spring-data-hadoop-1.0.1.RC1]$ ls -lrt
total 36
-rw-r--r-- 1 root root   861 Oct 11 01:32 readme.txt
-rw-r--r-- 1 root root 11357 Oct 11 01:32 license.txt
-rw-r--r-- 1 root root  1151 Mar  4 06:19 notice.txt
drwxr-xr-x 2 root root  4096 Mar 20 20:49 dist
drwxr-xr-x 4 root root  4096 Mar 20 20:49 docs
drwxr-xr-x 3 root root  4096 Mar 20 20:49 schema
drwxr-xr-x 2 root root  4096 Mar 20 20:49 samples
```

Please refer to the `readme.txt` and files within `docs/` directory to start using Spring Data Hadoop.

HVE (Hadoop Virtualization Extensions)

Hadoop was first designed for typically running on commodity hardware and native OS. Hadoop Virtualization Extensions were developed to enhance Hadoop running in cloud and virtualization environments.

Topology Awareness

Hadoop Virtualization Extensions (HVE) allow Hadoop clusters implemented on virtualized infrastructure full awareness of the topology on which they are running, thus enhancing the reliability and performance of these clusters.

HVE should be enabled in the following situations:

- When there is more than one Hadoop VM per physical host in virtualized environments
- When Datanodes and TaskTrackers exist in separate virtual machines in virtualized environments, so as to achieve graceful scaling of the compute component of the Hadoop cluster.
- When there is a topology layer between host and rack (e.g. chassis), which can affect the failure/locality group between hosts, in non-virtualized environments.

Elasticity

HVE Elastic Resource Extension enables the adaption of MapReduce tasks to changing resources on nodes/clusters where Hadoop clusters are deployed on virtualization by sharing resource with VMs from other clusters or applications.

Overview

Currently, the Hadoop resource model is static at the node level, assuming the node resources are not changed while the cluster is running. This design and implementation are based on an assumption that all cluster resources are dedicated for Hadoop MapReduce jobs, so they are fully available at all times. This assumption does not hold when users want to deploy multiple applications on the same cluster, e.g. deploying HBase and MapReduce on the same HDFS cluster. In particular, in an era of cloud computing, it is common for Hadoop clusters to be deployed on virtualization by sharing resource with VMs from other clusters or applications.

The HVE elastic resource feature addresses scenarios in which nodes' resources are possibly changed, so that scheduling of MapReduce tasks on these nodes can adapted to changing resources, as represented in the figure below.

With this feature, APIs (CLI and JMX interface) and script tools are provided to get/set map and reduce task slots on Hadoop cluster nodes for MR jobs.

Function List

Below are functionalities included in this elastic feature:

Table 3.1 Function List

Function	Description
Configuration	Enable/disable elastic resource feature on Hadoop cluster by specifying a configuration property when starting MR cluster.
List nodes' status	List the status of all the nodes or nodes specified by user, including its tracker_name, hostname, health status, slot number, etc.
Set map/reduce slot on specific node	Set map/reduce slot number to a node specified by user via CLI or JMX interface.
Set map/reduce slot on nodes in batch mode	Set map/reduce slot number on nodes specified by a node-slot mapping file.

Configuration

Elastic resource feature is disabled by default. To enable elastic resource, make the following changes to the Hadoop configuration.

In `mapred-site.xml`, add the following property to enable the elastic resource feature:

```
<property>
   <name>mapreduce.dynamic.resource.enabled</name>
   <value>true</value>
</property>
```

Also, specifying the node's map and reduce slot number in `mapred-site.xml` will also be the upper limit of node's slot resource. Any slot number update, if greater than this value, will be replaced with the value specified here. In following example, the maximum map slot number is 5 and maximum reduce slot number is 2.

Map:

```
   <property>
      <name>mapred.map.tasks</name>
      <value>5</value>
   </property>
```

Reduce:

```
   <property>
      <name>mapred.reduce.tasks</name>
      <value>2</value>
   </property>
```

Command Line Interface

List all CLIs of MRAdmin

```
hadoop mradmin
    [-refreshServiceAcl]
    [-refreshQueues]
    [-refreshUserToGroupsMappings]
    [-refreshSuperUserGroupsConfiguration]
    [-refreshNodes]
    [-listNodes]
    [-nodeStatus <tracker name>]
    [-setSlot <tracker name> <map | reduce> <slot number>]
    [-setSlotByHost <hosname> <map|reduce> <slot number>]
    [-help [cmd]]
```

List Nodes

```
hadoop mradmin -listNodes
```

Get Node Status

```
hadoop mradmin -nodeStatus <TRACKER-NAME>
```

Set Slot Number with Tracker_Name

Map:

```
hadoop mradmin -setSlot <TRACKER_NAME> map <NUM>
```

Reduce:

```
hadoop mradmin -setSlot <TRACKER_NAME> reduce <NUM>
```

Set Slot Number with Hostname

Map:

```
hadoop mradmin -setSlotByHost <HOSTNAME> map <NUM>
```

Reduce:

```
hadoop mradmin -setSlotByHost <HOSTNAME> reduce <NUM>
```

Updating Slots in Batch mode

Batch Updating Slots Script

Location: `${HADOOP_HOME}/bin/update-slots.sh`

The script will apply the slot updating operation in batch mode on nodes by resolving a slot description file.

Slot Description File

Location of this file is by default as below:

```
${HADOOP_HOME}/conf/slots
```

Also, you can also specify the location of this file in running this script tool:

Usage: `update-slots.sh  PATH_TO_SLOT_ FILE`

The slot description file format is as following:

```
hostname1    map_slots    reduce_slots
hostname2    map_slots    reduce_slots
hostname3    map_slots    reduce_slots
...
```

Below is an example:

```
hadoop@Hadoop:~ $ cat ${HADOOP_HOME}/conf/slots
Hadoop-01   2   2
Hadoop-02   1   1
```

Example

The setup of cluster with 3 nodes is as following:

Table 3.2 Cluster Setup

HostName	Role
Namenode	JobTracker / Namenode
Hadoop-01	TaskTracker / Datanode
Hadoop-02	TaskTracker / Datanode

1. CLI

a. listNodes:

```
hadoop@NameNode:~$ hadoop mradmin -listNodes
TaskTracker
Health UsedMap MaxMap UsedReduce MaxReduce
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
OK         0                2              0                 2
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
OK         0                1              0                 1
```

b. nodeStatus:

```
hadoop@NameNode:~$ hadoop mradmin -nodeStatus
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
{
  "failures" : 0,
  "slots" : {
    "map_slots" : 1,
    "reduce_slots" : 1,
    "reduce_slots_used" : 0,
    "map_slots_used" : 0
  },
  "tracker_name" :
"tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516",
  "hostname" : "Hadoop-01",
  "health" : "OK",
  "last_seen" : 1359570978699
}
hadoop@NameNode:~$ hadoop mradmin -nodeStatus
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
{
  "failures" : 0,
  "slots" : {
    "map_slots" : 2,
    "reduce_slots" : 2,
    "reduce_slots_used" : 0,
    "map_slots_used" : 0
  },
  "tracker_name" :
"tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722",
  "hostname" : "Hadoop-02",
  "health" : "OK",
  "last_seen" : 1359570988442
}
```

c. setSlot (map):

```
hadoop@NameNode:~$ hadoop mradmin -setSlot
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722 map
1
Set map slot of
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
successfully.
hadoop@NameNode:~$ hadoop mradmin -listNodes
TaskTracker
Health UsedMap MaxMap UsedReduce MaxReduce
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
OK         0               1              0                 2
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
OK         0               1              0                 1
```

d. setSlot (reduce):

```
hadoop@NameNode:~$ hadoop mradmin -setSlot
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
reduce 1
Set reduce slot of
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
successfully.
hadoop@NameNode:~$ hadoop mradmin -listNodes
TaskTracker
Health UsedMap MaxMap UsedReduce MaxReduce
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
OK          0              1              0                 1
```

```
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
OK         0              1              0                  1
```

e. setSlotByHost (map):

```
hadoop@NameNode:~$ hadoop mradmin -setSlotByHost Hadoop-01
map 1
Set map slot of
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
successfully.
hadoop@NameNode:~$ hadoop mradmin -listNodes
TaskTracker
Health UsedMap MaxMap UsedReduce MaxReduce
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
OK          0             2               0                 2
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
OK          0             1               0                 2
```

f. setSlotByHost (reduce):

```
hadoop@NameNode:~$ hadoop mradmin -setSlotByHost Hadoop-01
reduce 1
Set reduce slot of
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
successfully.
hadoop@NameNode:~$ hadoop mradmin -listNodes
TaskTracker
Health UsedMap MaxMap UsedReduce MaxReduce
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
OK          0              2              0                 2
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
OK          0              1              0                 1
```

2. Batch Updating Slots

a. Use the slot description file by default:

```
hadoop@NameNode:~/hadoop-1.2.0$ bin/update-slots.sh
Set map slot of
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
successfully.
Set reduce slot of
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
successfully.
Set map slot of
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
successfully.
Set reduce slot of
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
successfully.
hadoop@NameNode:~/hadoop-1.2.0$ hadoop mradmin -listNodes
```

```
TaskTracker
Health UsedMap MaxMap UsedReduce MaxReduce
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
OK          0              1             0                 1
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
OK          0              2             0                 2
```

b. Specify the slot description file:

```
hadoop@NameNode:~/hadoop-1.2.0$ bin/update-slots.sh
slot_desc_file
Set map slot of
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
successfully.
Set reduce slot of
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
successfully.
Set map slot of
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
successfully.
Set reduce slot of
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
successfully.
hadoop@NameNode:~/hadoop-1.2.0$ hadoop mradmin -listNodes
TaskTracker
Health UsedMap MaxMap UsedReduce MaxReduce
tracker_Hadoop-02:localhost.localdomain/127.0.0.1:35722
OK          0              1             0                 1
tracker_Hadoop-01:localhost.localdomain/127.0.0.1:48516
OK          0              1             0                 1
```

HDFS Rack Awareness

HDFS rack awareness is a key feature to achieve localized I/O (locality).

With respect to read and write separately, HDFS has:

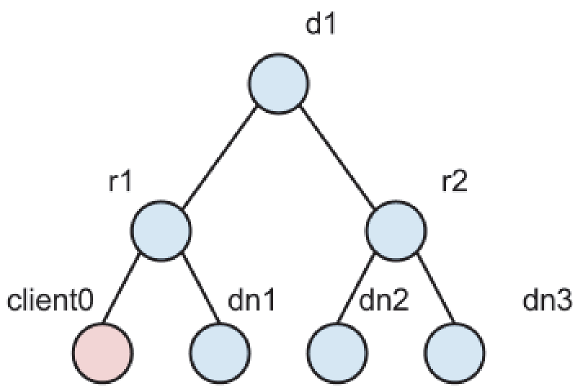
- BlockPlacementPolicy for write locality: namenode will look up network topology and construct a list of chosen nodes (pipeline) for a requesting a block to locate, based on algorithms provided by a BlockPlacementPolicy.
- Block pseudo distance sort for read locality: when reading a block, after obtaining all the located blocks, namenode sorts these located blocks based on their topological distance with client. The closer nodes get higher priority for read.

Both operations need to reference network topology, which is managed by the rack awareness feature. The rack awareness feature includes:

- A topology resolving framework: when datanodes register themselves on a namenode, that namenode will resolve their network location using their host name or ip, using DNSToSwitchMapping. This is a pluggable component that allows users to define their own topology based on their network layout. The most commonly used DNSToSwitchMapping is ScriptBasedMapping, which calls a shell script.
- An in-memory topology tree: all registered datanodes' network locations are kept in a topology tree.

Problem: Ignored off-cluster clients

The problem of the current implementation is that it do not support off-cluster clients. The figure below is an example of off-cluster clients:



In that figure, node **dn1** is a datanode and its network location is /d1/r1, and so on for **dn2** and **dn3**. Node **client0** is an off-cluster node, which means there is no datanode deployed on **client0**. In this case, **client0** has no chance to register itself in the topology tree of the namenode. Therefore both read and write operations select random nodes even though **dn1** is closer (more preferable) than either **dn2** or **dn3**.

This problem will cause performance issues in the following cases:

- When a mapreduce cluster is not exactly co-located: some mapreduce clusters share the same hdfs cluster with other mapreduce clusters, or in some cases a mapreduce cluster will cover several hdfs clusters. In those cases, a big portion of I/O will be off-cluster client operations which can not benefit from localized I/O.
- When a physical cluster is not dedicated to hadoop: a physical cluster may not be dedicated to hadoop and other supporting systems such as data loading tools may share the same cluster. In that case, the data loading tool can not benefit from localized I/O, even if the tool and hdfs shares the same rack/data center. The problem may even more common in virtualized environment.

Solution: Design

To tackle this problem, we changed the logic in block placement policy and block pseudo distance sort. We also resolved the network location of the client.

Resolving client location

Resolving the client location: we reused the framework that resolves datanodes. However, since we did not add client network locations into topology tree (as explained below), we have to cache client locations to avoid unnecessary resolve operations.

As a result, we introduced two LRU caches:

- A black list for those clients who have no valid location or whose locations do not share the same rack with any datanode.
- A white list opposite to the black list, for those clients who are not datanodes but share the same rack with at least one datanode.

Referring to the diagram of ignored off-cluster clients, the table below lists some examples of location cache.

Table 3.3 Location Cache Examples

HostName	Location	Cache
client1	d1/r1	white list
client2	d2/r1	black list
client4	null	black lis

The size of LRU cache is configurable so you can limit the memory usage of namenode.

Block placement policy

The tables below demonstrate how the BlockPlacementPolicy has been changed to support non-datanode clients.

Table 3.4 Former block placement algorithm

Replica	Rule
1	Client Local
2	Random node whose rack is different from replica 1
3	Random node who share the same rack with replica 2
>=4	Random node

New Block Placement Algorithm:

Table 3.5 Changed block placement algorithm

Replica	Rule
1	Client Local if client is datanode, or random node who shares the same rack with client if client is not a datanode
2	Random node whose rack is different from replica 1

Table 3.5 Changed block placement algorithm

Replica	Rule
3	Random node who shares the same rack with replica 2
>=4	Random node

Usage

The client rack aware feature is disabled by default. To enable, add the following to the `hdfs-site.xml` file:

```
<properties>
   <property>
      <name>dfs.rackawareness.with.client</name>
      <value>true</value>
   </property>
</properties>
<properties>
   <property>
      <name>dfs.rackawareness.with.client.blacklist.size</name>
      <description>Black list size of client cache, 5000 by
      default.</description>
      <value>5000</value>
   </property>
</properties>
<properties>
   <property>
      <name>dfs.rackawareness.with.client.cache.size</name>
      <description>White list size of client cache, best set it
      equals the size of cluster. 2000 by default.</description>
      <value>2000</value>
   </property>
</properties>
```

Note that you need to restart DFS after changing the configuration.

Vaidya

Vaidya Overview

Vaidya is a diagnostic tool installed with PHD for Map/Reduce jobs. After a job is executed successfully, it uses a job history log and job configuration information to identify any performance or scalability problems with the job. Upon execution, it provides a job analysis report indicating specific problems with the job along with the remedy to correct them. The report element includes, "rule title", "rule description", "rule importance", "rule severity", "reference details" and "remedy/prescription" to rectify the problem. The "rule severity", is a product of rule impact and the rule importance.

Note: The Vaidya tool does *not* analyze failed jobs either for performance or scalability problems nor for the reasons of failure.

The Vaidya tool includes diagnostic rules (also referred to as "tests") where each rule analyzes a specific problem with the M/R job. Diagnostic rule is written as a Java class and captures the logic of how to detect a specific problem condition with the M/R job. Each diagnostic rule takes job history log and job configuration information provided to it using a standard structured interface. The standard interface allows administrators and developers to independently add more diagnostic rules in the Vaidya tool.

Note that Vaidya is installed together with PHD and is by default enabled. No additional installation and configuration needed.

Installing Vaidya Files

By default, Vaidya files are installed at:

- The Vaidya JAR library is installed into `/usr/lib/gphd/hadoop-mapreduce/`
- The Vaidya default test configuration file is installed into `/etc/gphd/hadoop/conf/`

Enabling and Disabling Vaidya

By default, Vaidya is enabled after installation, there is normally no need to enable it manually.

Enabling Vaidya

In cases where Vaidya is not enabled and you want to enable it explicitly:

On the job tracker node, go to the PHD configuration folder (by default, `/etc/gphd/hadoop/conf`), and add the following lines into the file `mapred-site.xml`.

mapred-site.xml

```
<property>
    <name>mapreduce.vaidya.enabled</name>
    <value>true</value>
```

```
</property>
<property>
   <name>mapreduce.vaidya.jarfiles</name>
   <value>/usr/lib/gphd/hadoop-mapreduce/hadoop-vaidya-default.
   jar</value>
</property>
<property>
   <name>mapreduce.vaidya.testconf.file</name>
   <value>/etc/gphd/hadoop/conf/postex_diagnosis_tests.xml</val
   ue>
</property>
```

Disabling Vaidya

To disable Vaidya:

Set the property `mapreduce.vaidya.enabled` value to be `false`, or remove these lines from `mapred-site.xml`.

Note:

- The value of property `mapreduce.vaidya.enabled` should be changed to point to the correct jar file you installed. By default, this is `/usr/lib/gphd/hadoop/contrib/vaidya/hadoop-vaidya-<HADOOP_PHD_VERSION>.jar` or `/usr/lib/gphd/hadoop-mapreduce/hadoop-vaidya-<HADOOP_PHD_VERSION>.jar`.
- Once you edit the xml file, restart the job history server service to ensure the change takes effect.

Using Vaidya to Analyze Jobs

To use Vaidya with PHD 1.0.x and to ensure your job history server service is running:

1. Successfully run a map-reduce job for Vaidya to analyze. Refer to *Pivotal HD Enterprise 1.0 Installation and Administrator Guide* for instructions about how to run map-reduce job with PHD.
2. Ensure your job history server service is running.
3. Open the following URL in a web browser:
   `http://<historyserver_host>:<historyserver_port>/jobhistory`
   Where:
   - *`<historyserver_host>`* refers to the host name or IP address of the machine where you run job history server service.
   - *`<historyserver_port>`* refers to the HTTP port job history server web where the UI listens. By default, this value is 19888. Your browser should show you the job history server UI page.
4. You will see a list of jobs that have run, including the most recent job. Click the job id of any job in this list, and you should see the detailed information of the job.

5. On the left side of the navigation area, there should be a link called **Vaidya report** under the navigation item **Job**. Click the **Vaidya report** link and Vaidya will analyze the job for you and show a report.

Vaidya Configuration Rules

After you installed Vaidya with PHD, rules configuration is installed as a `postex_diagnosis_tests.xml` XML file, here: `/etc/gphd/hadoop/conf/`

You can find all rules to be run on a selected job in that XML file, where each rule is defined as an XML `PostExPerformanceDiagnosisTests/DiagnosticTest` element, for example:

A rule in `postex_diagnosis_tests.xml`

```
<DiagnosticTest>
   <Title><![CDATA[Balanced Reduce Partitioning]]></Title>
   <ClassName>
<![CDATA[org.apache.hadoop.vaidya.postexdiagnosis.tests.BalancedRe
ducePartitioning]]></ClassName>
   <Description><![CDATA[This rule tests as to how well the input
   to reduce tasks is balanced]]></Description>
   <Importance><![CDATA[High]]></Importance>
   <SuccessThreshold><![CDATA[0.40]]></SuccessThreshold>
   <Prescription><![CDATA[advice]]></Prescription>
   <InputElement>

      <PercentReduceRecords><![CDATA[85]]></PercentReduceRecords>
   </InputElement>

</DiagnosticTest>
```

The `Title` and `Description` elements provide a brief summary about what this rule is doing.

By editing `postex_diagnosis_tests.xml`, you can configure the rules.

Notes:

- Remember to backup original configuration file before editing the configuration file, invalid xml config file may cause Vaidya behavior incorrectly.
- Before you start editing rules, you should have background knowledge about XML syntax and how XML represents data (for example, what the `CDATA` element represents).

Disabling a Rule

Comment out or remove the whole DiagnosticTest element.

Changing the Importance of a Rule

Importance indicates how relatively important a rule is, relative to other rules in the same set. You can change the importance value by editing Importance element in the XML file. A level served as a factor which is multiplied to impact value returned by each rule.

There are three values valid for this attribute: Low, Medium and High; their corresponding values are: 0.33, 0.66 and 0.99.

In the displayed Vaidya report, there is a value named Severity for each rule. A severity level is the result of multiplying the impact value (returned by rule) and the importance factor (defined in XML file).

For example, a rule returns impact of 0.5, its importance is marked as Medium, then its severity is 0.5 * 0.66 = 0.33.

Changing Success Threshold

Each rule calculates a value between 0 and 1 (inclusively) to indicate how healthy a job is according to the given rule, this value is called impact. The smaller the impact is (that is, closer to 0), the healthier the job is.

To give a more straight forward result, you can set a threshold for each rule, therefore a rule whose impact value is larger than the threshold will be marked as "failed", otherwise, it is marked as "passed".

Note that threshold is compared with impact value, rather than severity (which means make a rule less important will not make a failed rule succeed).

You can change the threshold value by editing the SuccessThreshold element in the XML file.

Changing Input Parameters

Some rules may need additional input parameters to complete their logic. You can specify additional parameters by editing/adding elements under the InputElement element of each rule.

Other

For a full explanation and instruction about the meaning of each XML element and how to change them, refer the Apache's Official `Vaidya Guide` for more information.

Adding a New Rule

A Vaidya rule consists of the following two parts:

- A java class that consists of the logic of the rule
- A paragraph of XML in the configuration file

Creating a Java Binary for a New Rule

Important: This section assumes a working knowledge of how to write, compile, and package Java code.

1. From where you installed PHD, download the correct `hadoop-vaidya-<HADOOP_PHD_VERSION>.jar` file (which you specified in `mapred-site.xml`) to your development machine, if you plan on writing Java code on another machine than the one where you installed PHD (This is a typical case).

2. Create a java file with an IDE or editor, which defines a class that extends the `org.apache.hadoop.vaidya.DiagnosticTest` class:

myrule.java

```
package com.greenplum.vaidya.rules;

import org.apache.hadoop.vaidya.DiagnosticTest;

import org.apache.hadoop.vaidya.statistics.job.JobStatistics;

public class MyRule extends DiagnosticTest {
   @Override
   public String getReferenceDetails() {
      return "";
   }
   @Override
   public String getPrescription() {
      return "";
   }
   @Override
   public double evaluate(JobStatistics jobStatistics) {
      return 0.5;
   }
}
```

3. Edit the three methods `getReferenceDetails`, `getPrescription` and `evaluate` to construct the logic.
   - `evaluate` method should return a **double** value between 0.0 and 1.0 and represents the impact as the analysis result.
   - `getPrescription` method should return some text providing user suggestions/remedies about how to optimize your Map/Reduce configuration accordingly.
   - `getReferenceDetails` method should return some text indicating the meaningful counters and their values which can help you to diagnose your Map/Reduce configuration accordingly.

4. Compile the java class and package compiled class to a `jar` file, e.g., `myrule.jar`. Note that you need to put the Vaidya jar file you just downloaded into your class path to make your code compile.

Creating XML Configuration For a New Rule

Add a `DiagnosticTest` element into the `postex_diagnosis_tests.xml` file (the file you set in `mapred-site.xml` file), according to the sample given in the configuration part. Ensure the value of `ClassName` element is set to be the full class name of the java rule class you just created.

Deploying files

1. Upload the packaged jar file (`myrule.jar` for example) to the node where you installed PHD job tracker, and store it in a folder where hadoop service has the permission to read and load it. We recommend you place it under `/usr/lib/gphd/hadoop-mapreduce/lib/`
2. Edit `mapred-site.xml`, append the jar file you just uploaded to the `mapred.vaidya.jar.file` or `mapreduce.vaidya.jarfiles` property value, for example:

```
mapred-site.xml

<property>
   <name>mapreduce.vaidya.jarfiles</name>
   <value>/usr/lib/gphd/hadoop-mapreduce/hadoop-vaidya-2.0.2-alpha
   -gphd-2.0.1.0.jar:/usr/lib/gphd/hadoop-mapreduce/lib/myrule.jar
   </value>
</property>
```

Important:

- Do not remove the default Vaidya jar file from this property, Vaidya needs this property to load basic Vaidya classes to make it run.
- Multiple jar files are separated by different separator characters on different platforms. On the Linux/Unix platform, the "`:`" character should be used. You can look at the`File.pathSeparator` attribute of your java platform to ensure it.

3. To make your settings take affect, restart job history server service.
4. Run your Vaidya analysis again, you should see your rules run and results appear in the report.

DataLoader

See the *Greenplum DataLoader Installation and User Guide* for detailed information.

4. Unified Storage System (USS)

This chapter installation, configuration, usage, and troubleshooting about the USS service.

Overview

USS is a service on Pivotal HD that provides a unified namespace view of data across multiple filesystems.

USS enables users to access data across multiple filesystems, without copying the data to and from HDFS. It is the underlying technology for enabling Data Tiering/Retention and Migration from one Hadoop distribution to another.

USS is implemented as a 'pseudo Hadoop File System' that delegates File System operations directed at it to other filesystems in a HDFS-like way. It mounts multiple filesystems and maintains a centralized view of the mount points, which are accessible through the URI scheme of `uss://`. It relies on a catalog service for managing metdata about mount points and delegated filesystems.

Versions

This section contains information about USS version 0.3.0, which is compatible with Pivotal HD 1.0.1 (Hadoop version 2.0.x).

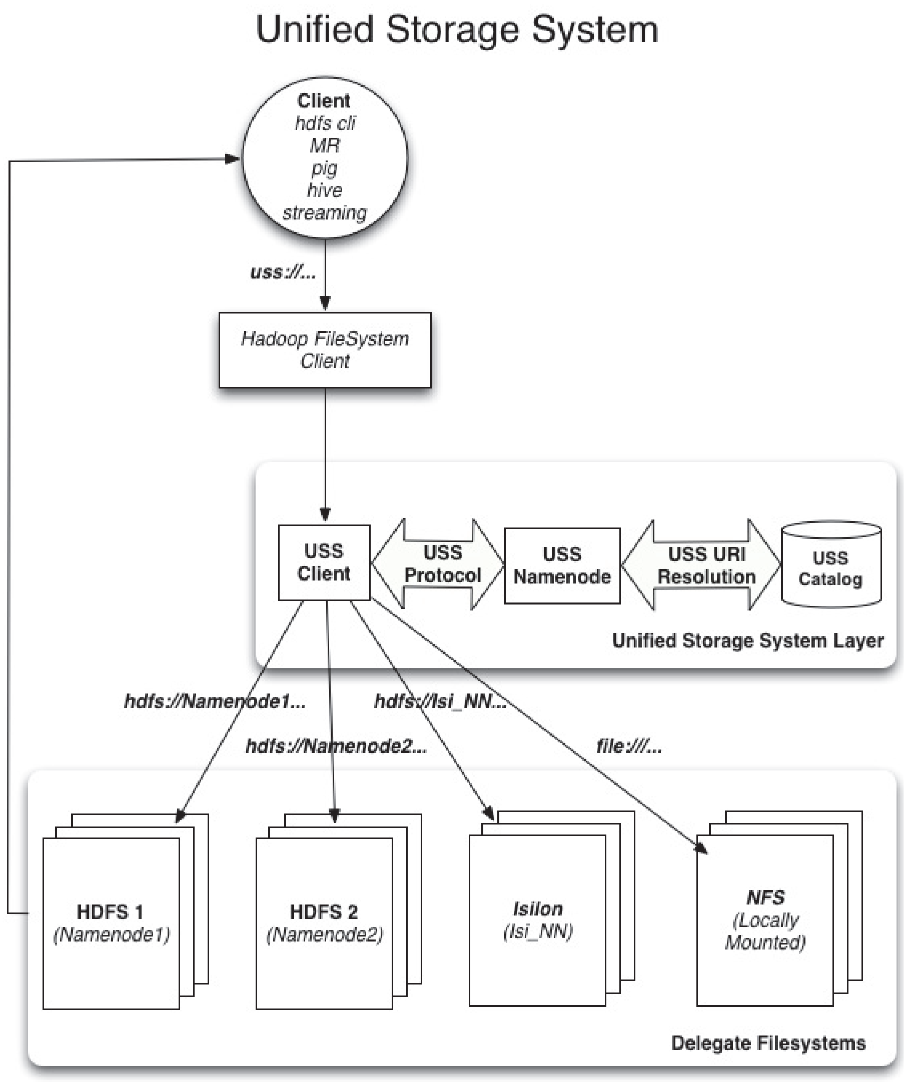
Supported Features

USSv 0.3.0 supports the following features:

Table 4.1 Supported Features

Feature	Description
Compatibility	Pivotal HD 1.0.1 (Hadoop 2.0.x)
Supported Delegate File Systems	• HDFS 2.x • Isilon (OneFS) • LocalFS (Mount point target paths assumed to be present on all datanodes/nodemanagers) • NFS (Mounted on all datanodes/nodemanagers) • FTP (with known issues)
Modes of access/interfaces	• HDFS CLI • Native Java Map-Reduce Jobs • Hadoop Streaming • Pig • Hive (External tables)
Installation/Configuration	Fully managed through Pivotal HD Manager 1.0.1
Catalog	Postgres database

Architecture



USS is implemented as a 'pseudo Hadoop FileSystem' that:

- Accepts Hadoop File System requests containing **USS URI**s
- Extracts **USS Mount Points** from USS URIs
- Resolves USS Mount Points to their **Actual Paths** on supported **Delegate File Systems**
- Delegates the File System requests to the Delegate File Systems

Refer to "Terminology" on page 80 for a description of these terms.

Prerequisites

- Platform

  RHEL 6.1/6.2 64Bit or CentOS 6.1/6.2 64Bit
- Pivotal HD Hadoop

  Pivotal HD Version 2.0.2-alpha-gphd-2.0.1

  Pivotal HD is available through via the `EMC Download Center` (paid version) or the `Pivotal HD product page` (Community Edition). Please contact your local Pivotal HD Support if you need help downloading Pivotal HD.
- For USS to support a storage system, it must either be a Hadoop File System, or it must have a Hadoop File System compatible implementation.
- Installation/Setup/Configuration/Maintenance of delegate file systems is beyond the purview of USS. USS assumes that these file systems are available for serving requests.
- Oracle Java 1.6 JDK needs to be installed in all nodes in the cluster, including the admin node. Use RPM install to setup necessary paths for JDK on the machine. Download and execute self-executing RPM binary (for example: `jdk-6u43-linux-x64-rpm.bin`) from the Oracle site `http://www.oracle.com/technetwork/java/javase/downloads/jdk6downloads-1902814.html` after accepting the license.

Getting Started

USS (v0.3.0) is part of Pivotal HD 1.0.1. USS is distributed as a set of three RPM packages that are part of the `PHDTools-1.0.1.0` tarball. You can download the PHDTools tarball d-2.0.1from the `EMC Download Center` (paid version) or the `Pivotal HD product page` (Community Edition).

Architecturally, USS has the following four components:

USS Namenode (Server)

The USS Namenode is a stateless server-side component. It is implemented as a Hadoop RPC server, that is capable of interacting with a catalog system to delegate file system calls to appropriate underlying file systems. It is expected to run on a dedicated server. In this alpha version, this runs as a single server, with no HA and failover. It is designed to be the central place where all file system calls converge, before they are delegated to a multitude of underlying file systems. It abstracts datasets on underlying file systems as mount-points. It stores the mapping between mount-points and the directories that they point to in the USS catalog. The USS Namenode does not store any state, its state is contained within the USS Catalog.

USS Catalog

The USS Catalog is a metadata store for the USS Namenode. In this version, the USS catalog is contained in a postgres database. USS ships scripts that can help users manage the USS Catalog. The USS RPMs do not install the postgres database to be used as the USS catalog. It is expected that users have a postgres database ready. However, if users use Pivotal Command Center to deploy USS, it can deploy a postgres database as the USS Catalog.

USS Client

The USS Client component is expected to be present on all nodes in a Pivotal HD cluster. The USS client accepts a FileSystem request (containing a mount-point), maps the request to the appropriate File System and path, by making an RPC request to the USS Namenode.

USS Admin

The USS Admin hosts allow administrators to update the USS Catalog. This is an optional role. If not configured, users can use the USS Namenode as the admin node.

USS Directory Contents

The contents of USS 0.3.0 within the `PHDTools-1.0.1.0` tarball are located here:

```
PHDTools-1.0.1-5/uss/
|--rpm
   |--uss-0.3.0-5.noarch.rpm
   |--uss-0.3.0-5.noarch.rpm.md5
   |--uss_catalog-0.3.0-5.noarch.rpm
   |--uss_catalog-0.3.0-5.noarch.rpm.md5
   |--uss_namenode-0.3.0-5.noarch.rpm
   |--uss_namenode-0.3.0-5.noarch.rpm.md5
```

USS RPM Packages

USS is distributed as a set of three RPM packages:

Package Descriptions:

Table 4.2 rpm package descriptions

PRPM Package Name	Description	Dependencies	Hosts to install on
`uss-0.3.0-5.noarch.rpm`	The base USS package. It installs the USS library and required configuration files.	Pivotal HD hadoop package (version 2.0.2-alpha-gphd-2.0.1 and above)	All hosts using USS (agent, namenode, admin and catalog hosts)

Table 4.2 rpm package descriptions

PRPM Package Name	Description	Dependencies	Hosts to install on
`uss_catalog-0.3.0-5.noarch.rpm`	This package installs the scripts and configuration files required to interact with and administer the USS Catalog.	uss (version >= 0.3.0)	USS Admin hosts. Installed on the USS Namenode host by default as the uss_namenode RPM package has a dependency on it.
`uss_namenode-0.3.0-5.noarch.rpm`	This is the USS Namenode package. It installs the USS namenode service daemon and its runtime configuration.	uss_catalog package (version >= 0.3.0).	USS Namenode host

Package Contents

uss-0.3.0-5.noarch.rpm

Table 4.3 `uss-0.3.0-5.noarch.rpm`

File	Description
`/usr/lib/gphd/uss/uss-0.3.0.jar`	The USS library
`/usr/lib/gphd/uss/uss-javadoc-0.3.0.jar`	The USS javadocs
`/etc/gphd/uss/conf/commons-logging.properties and /etc/gphd/uss/log4j.properties`	USS log configuration files
`/etc/gphd/uss/conf/uss-client-site.xml`	USS client side configuration file
`/etc/gphd/uss/conf/uss-nn-site.xml`	USS Namenode configuration file
`/etc/gphd/uss/conf/uss-catalog.xml`	USS Catalog configuration file
`/etc/gphd/uss/conf/uss-env.sh`	USS Client side environment setup script
`/usr/lib/gphd/uss/docs/README.txt`	The USS installation, configuration and usage document.
`/usr/lib/gphd/uss/docs/CHANGES.txt`	The USS changelog.
`/usr/lib/gphd/uss/docs/NOTICE.txt`	The USS notice document.
`/usr/lib/gphd/uss/docs/README.txt`	The USS license document.

uss_catalog-0.3.0-5.noarch.rpm

Table 4.4 `uss_catalog-0.3.0-5.noarch.rpm`

File	Description
`/usr/bin/uss/`	This is the USS CLI to help admins/users to interact with the USS catalog.
`/usr/lib/gphd/uss/migration`	Contains utilities for migrating the USS catalog database.
`/usr/lib/gphd/uss/lib`	Contains the java libraries required to access the USS catalog.
`/usr/lib/gphd/uss/libexec`	Contains USS utility scripts.
`/etc/default/uss`	USS Catalog and Namenode runtime configuration script.
`/var/log//gphd/uss-catalog`	Log directory for USS admin commands

uss_namenode-0.3.0-5.noarch.rpm

Table 4.5 uss_namenode-0.3.0-5.noarch.rpm

File	Description
`/usr/sbin/uss-namenode-daemon`	The USS Namenode daemon script.
`/etc/rc.d/init.d/uss-namenode`	USS Namenode service script.

Table 4.5 uss_namenode-0.3.0-5.noarch.rpm

File	Description
`/var/log/gphd/uss`	USS Namenode log directory.
`/var/log//gphd/uss-namenode`	Log directory for USS Namenode

Installing USS

With Pivotal HD 1.0.1 users can install USS on their clusters using Pivotal HD Manager. This is the recommended way to install USS. For more information on installing USS using Pivotal HD Manager, please refer to the *Pivotal HD Manager 1.0 Installation and Administrator Guide* for details.

Configuring USS

The following sections describe the USS configurations in Pivotal HD 1.0.1. Pivotal HD Manager sets up all these configurations and no user action is necessary.

Enabling USS on your Pivotal HD Cluster

To enable USS, the following properties are set in `/etc/gphd/hadoop/conf/core-site.xml`.

```
!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
   <name>fs.uss.impl</name>
   <value>com.greenplum.hd.uss.USSProtocolTranslator</value>
   <description>The FileSystem for uss: uris.</description>
</property>
<property>
   <name>mapred.outdir.resolverClass</name>
   <value>com.greenplum.hd.uss.USSPathResolver</value>
   <description>The Class to use to resolve Output Directories
   for mapreduce jobs. Must implement
   org.apache.hadoop.mapred.PathResolver.</description>
</property>
</configuration>
```

Where:

- **`fs.uss.impl`**—USS is implemented as a pseudo hadoop file system. USS URI's use the scheme `uss`. Hadoop compatible file systems set the property `fs.<scheme>.impl` to the class that implements the file system. The class that implements the USS pseudo file system is `com.greenplum.hd.uss.USSProtocolTranslator`.

- **`mapred.outdir.resolverClass`**—If the output directory of a mapreduce job is set to a USS URI, hadoop uses this property to refer to a class that is capable of resolving the USS URI to a 'real' URI on an underlying file system. USS provides the class `com.greenplum.hd.uss.USSPathResolver` to resolve USS URIs

USS Namenode Configuration

These are the server-side configurations for USS.

```
uss-nn-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
   <property>
      <name>uss.namenode.address</name>
      <value>uss://USS_NAMENODE_HOST:16040</value>
      <description>Uri for USS Namenode.</description>
   </property>

   <property>
      <name>uss.catalog.type</name>
      <value>db</value>
      <description>Type of catalog to use.</description>
   </property>

   <property>
      <name>uss.db.name</name>
      <value>dbApplicationContext.xml</value>
      <description>Settings for postgres catalog.</description>
   </property>
</configuration>
```

Where:

- **`uss.namenode.address`**—The fully-qualified URI of the USS Namenode. Specifies the hostname and port on which the USS Namenode listens for requests.
- **`uss.catalog.type`**—Type of catalog to use. Defaults to db. We do not recommend changing this property.
- **`uss.db.name`**—Settings for postgres catalog. Defaults to `dbApplicationContext.xml`. We do not recommend changing this property.

USS Client Configuration

These are the client-side configurations for USS.

```
uss-client-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

   <property>
      <name>fs.uss.name</name>
      <value>uss://USS_NAMENODE_HOST:16040</value>
      <description>Fully qualified URI of the USS
      Namenode</description>
   </property>

   <property>
      <name>uss.tmp.fs</name>
      <value>hdfs://NAMENODE_HOST:9000</value>
      <description>Filesystem that hosts the temporary directory
      for map-reduce jobs.</description>
   </property>

</configuration>
```

Where:

- **`fs.uss.name`**—This property specifies the hostname and port of the USS Namenode on the client side. By default, it is set to uss://localhost:16040/. If the client is a non-USS Namenode host, then this property can be used to specify the hostname and port of the USS Namenode on the client.
- **`uss.tmp.fs`**—This property indicates the location that USS uses to store temporary data. All Hadoop-compatible file systems require a location to store temporary. Hadoop uses this location for creating the staging area for map-reduce jobs (.staging directory) as well as to store the .Trash directory. As USS is implemented as a pseudo Hadoop file system, it also requires a location for storing temporary data. However, as there is no storage/file system tied to USS, USS stores temporary data on the file system specified against this parameter. Users could choose to set this to a filesystem of their choice, as long as the file system is compatible with Hadoop (i.e. it has an HDFS implementation available). This file system should be available for USS to store temporary data. It could either be one of the underlying file systems, or a dedicated file system for storing temporary data.

USS Catalog Configuration

The USS Namenode as well as the USS CLI uses the configuration file `/etc/gphd/uss/conf/uss.properties` to read connection properties for the USS Catalog. If you wish to modify the USS Catalog Database URI, please update the property uss.db.url from this file. The value for this property is of the form:

```
uss.properties - uss.db.url
uss.db.url=jdbc:postgresql://[USS_CATALOG_DB_HOST]:[USS_CATA
LOG_DB_PORT]/usscatalog
```

We recommend that users only make sure that the host and port of the postgres server designated to be the USS Catalog is correct. Do not modify any other property from this file.

USS Catalog Administration - Using the USS CLI

USS provides a Command Line Interface for administrators to manage the USS catalog from the USS Admin hosts. The purpose of this tool is to provide a central/consistent way to manage the USS Catalog. This section describes the usage of the USS Admin CLI. Currently, USS Catalog resides in a postgres database and contains mainly USS Mount Point information. It is envisioned that the catalog will be enriched further.

Basics

- The *USS Catalog Management Tool* is accessed via the command: `uss`. This is a script located at `/usr/bin/uss`
- All nodes part of the `USS Admin` role have this command available to them. It is a part of the `uss_catalog` RPM package.
- These nodes need to have network reachability to the catalog (node on which the postgres database is installed)

Usage:

```
$ which uss
/usr/bin/uss

$ uss --help
 usage: uss admin|list|classpath|-h|--help [OPTIONS]
    -h,--help       show this help text
    admin <args>      perform USS Admin actions.
            Try 'uss admin -h|--help' to list the available
            admin options.
    classpath       display the java classpath for USS.
    list        list all USS mount points.
```

General Operations

Operations which do not inherently modify the catalog and can be used freely by all cluster users.

List existing mount points

Displays Mount Point Name, Mount Point ID and Mount Point Target URI for all mount points currently defined in the catalog.

```
$ uss list
click_data (1) > hdfs://NAMENODE_HOST1:8020/data/website/clicks
```

```
conv_data (2) > hdfs://NAMENODE_HOST2:8020/data/website/conversions
view_data (3) > file:///data/website/views
```

USS Classpath

General utility to view the java classpath used by USS.

USS Admin Operations

This encompasses the admin level operations that actually modify the catalog. It is encouraged that these operations be restricter to just a few `admin` users.

- Initialize USS Catalog
- Add a New Mount Point
- Delete an Existing Mount Point

USS Admin Usage

```
$ uss admin --help
usage: uss admin
 -a,--add                        Add a new mount point to the USS
                                 catalog
 -d,--delete                     Delete an existing mount point from
                                 the USS catalog
-h,--help                        Print help for this application
 -i,--initialize                 Setup the database for use with USS
 -l,--list                       List all mount points defined in
                                 the USS catalog
 -n,--mount-point-name <arg>     Name of the mount point to be added
                                 or deleted
-t,--mount-point-target <arg>    Target (full URI) of the mount
                                 point to be added
```

Initialize USS Catalog

- Create the `usscatalog` database on the host which has the postgres installation to use as the USS Catalog Database
- Create the `usscatalog` database owner
- Create the `usscatalog` database DDL
- Add default (as well as sample) data to the database
- Note: USS uses flyway to manage schema versioning and migration.
- Note: Pivotal HD Manager initializes the USS catalog database using this command automatically during cluster deployment.

  ```
  $ uss admin --initialize
  ```

Add a New Mount Point

Accepts the mount point name and mount point target URI as user input and adds a new mountpoint to the catalog

```
$ uss admin --add --mount-point-name NewMountPoint
--mount-point-target hdfs://NAMENODE1:8020/newPath
$ uss list
click_data (1)  >   hdfs://NAMENODE1:8020/data/website/clicks
conv_data (2)   >   hdfs://NAMENODE2:8020/data/website/conversions
view_data (3)   >   file:///data/website/views
NewMountPoint (5)   >   hdfs://NAMENODE1:8020/newPath
```

Delete an Existing Mount Point

Accepts the mount point name as user input and deletes the mount point from the catalog (if it exists).

```
$ uss admin --delete --mount-point-name NewMountPoint
```

```
$ uss list
click_data (1)  > hdfs://namenode1.mycompany.com:8020/data/website/clicks
conv_data (2)   > hdfs://namenode2.mycompany.com:8020/data/website/conversions
view_data (3)   >    file:///data/website/views
```

Using USS

Setting Up the Environment

Once USS is configured, users can access it by using the Hadoop Filesystem CLI or Map-reduce jobs. However, clients also need to set some environment variables prior to using USS. These variables make the USS library available in the classpath for hadoop commands. Pivotal HD Manager adds these variables to /etc/gphd/hadoop/conf/hadoop-env.sh by default, so users don't have to add them manually. These environment settings are listed below:

```
USS Environment Settings
$ export
HADOOP_CLASSPATH=/usr/lib/gphd/uss/uss-0.3.0.jar:/etc/gphd/uss/conf/
$ export HADOOP_USER_CLASSPATH_FIRST=true
```

The above environment settings have also been captured in the client side environment setup script `/etc/gphd/uss/conf/uss-env.sh`. Users may also edit this file and execute it before using USS.

Using USS Paths in map-reduce jobs

If users wish to use USS paths in map-reduce jobs, then they need one more extra configuration step. When the nodemanagers try to access USS paths, they need the USS library to resolve these paths. As a result, users need to add the same settings mentioned above in the `/etc/gphd/hadoop/conf/hadoop-env.sh` script and then restart the nodemanager on all nodes. To do this, add the following lines to `/etc/gphd/hadoop/conf/hadoop-env.sh` on all nodemanager nodes -

```
/etc/gphd/hadoop/conf/hadoop-env.sh

export HADOOP_CLASSPATH=/usr/lib/gphd/uss/uss-0.2.0.jar:/etc/gphd/uss/conf/

export HADOOP_USER_CLASSPATH_FIRST=true
```

Testing USS

Execute hadoop filesystem commands (using the Hadoop CLI) or mapreduce jobs to test

To test USS, users can execute some hadoop filesystem CLI commands, mapreduce jobs, hadoop streaming jobs, pig scripts or hive scripts using mount-points defined in the catalog.

USS URIs

A USS URI is a URI that USS can understand and resolve. It does not point to a 'real' location on a FileSystem, but contains an index into a Catalog that can be used to lookup for the actual URI on an underlying 'Delegate FileSystem'. A USS URI has the form:

```
uss://<USSNameNode Host>:<USSNameNode Port>/<Mount-Point>/<Sub-path under
the mount-point>
```

Examples using USS URIs in Hadoop Filesystem CLI

```
# Prepare input directory

hadoop fs -mkdir /tmp/test_uss_input

# Copy data from /etc/passwd

hadoop fs -copyFromLocal /etc/passwd /tmp/test_uss_input

# Add mount-point for input. Do this on the
uss-admin/uss-namenode/uss-catalog host

$ uss admin --add --mount-point-name input_mount_point --mount-point-target
hdfs://NAMENODE_HOST:8020/tmp/test_uss_input

# Add mount-point for output. Do this on the
uss-admin/uss-namenode/uss-catalog host

$ uss admin --add --mount-point-name output_mount_point
--mount-point-target hdfs://NAMENODE_HOST:8020/tmp/test_uss_output
```

```
# List contents of the input directory using USS

$ hadoop fs -ls uss://USS_NAMENODE:16040/input_mount_point

Found 1 items

-rw-r--r--   1 user wheel       1366 2012-08-17 10:08
hdfs://NAMENODE_HOST:8020/tmp/test_uss_input/passwd
```

Examples using USS URIs in Mapreduce jobs

```
$ hadoop jar
/usr/lib/gphd/hadoop-mapreduce/hadoop-mapreduce-examples-2.0.2-alpha-gphd-2
.0.1.0.jar wordcount uss://USS_NAMENODE_HOST:16040/input_mount_point
uss://USS_NAMENODE_HOST:16040/output_mount_point/uss_mr_output
```

Examples using USS URIs in Pig Scripts

```
\# pig script

$ cat uss.pig

A = load 'uss://USS_NAMENODE_HOST:16040/input_mount_point';

B = foreach A generate flatten(TOKENIZE((chararray)$0, ':')) as word;

C = filter B by word matches '

w+';

D = group C by word;

E = foreach D generate COUNT(C), group;

STORE E into
'uss://USS_NAMENODE_HOST:16040/output_mount_point/uss_pig_output';

$ pig -Dpig.additional.jars=/usr/lib/gphd/uss/uss-0.3.0.jar uss.pig //
Execute pig script, by adding the uss jar as an additional jar to include.
```

Examples using USS URIs in Hive Scripts

```
$ cat uss-hive.sql

-- creates an external table with location pointed to by a USS URI.


DROP TABLE test_uss_external;


CREATE EXTERNAL TABLE test_uss_external (testcol1 STRING, testcol2 STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ':'

LINES TERMINATED BY '\n'

STORED AS TEXTFILE
```

```
LOCATION 'uss://USS_NAMENODE_HOST:16040/input_mount_point';

SELECT * FROM test_uss_external;


$ hive -f uss-hive.sql
```

Examples using USS URIs in Streaming jobs

```
\# streaming mapper
$ cat uss-streaming-mapper.py
#!/usr/bin/env python

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for USSStreamingReducer.py
        #
        print '%s\t%s' % (word, 1)

\# streaming reducer
$ cat uss-streaming-reducer.py
#!/usr/bin/env python

from operator import itemgetter
import sys
```

```
current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output by key
    # (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
        # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)

\# run
```

```
$ hadoop jar
/usr/lib/gphd/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.0.2-alpha-gp
hd-2.0.1.jar \
-input uss://USS_NAMENODE_HOST:16040/input_mount_point \
-output
uss://USS_NAMENODE_HOST:16040/output_mount_point/uss_streaming_output \
-mapper uss-streaming-mapper.py
-reducer uss-streaming-reducer.py
```

USS Namenode Service

USS exposes the USS Namenode service. This service listens for Hadoop RPC requests from clients at a specific port (configurable via uss.namenode.address in uss-nn-site.xml on the USS Namenode host). This service can be started using service script `/etc/rc.d/init.d/uss-namenode`.

Starting the service

```
$ service uss-namenode start
```

Stopping the service

```
$ service uss-namenode stop
```

Restarting the service

```
service uss-namenode restart
```

Checking the service status

```
service uss-namenode status
```

This service internally uses the USS daemon script installed at `/usr/sbin/uss-namenode-daemon` to perform the desired function.

It uses the following runtime files:

- **`/var/run/uss-namenode.pid`**–USS Namenode service pid file.
- **`/var/lock/subsys/uss-namenode`**–USS Namenode service lock file.
- **`/var/log//gphd/uss-namenode/uss-namenode.log`**–USS Namenode log file

Uninstallation

USS can be uninstalled by erasing the USS RPM packages from the hosts on which it was installed.

```
$ yum erase uss_namenode -y
$ yum erase uss -y
```

Known limitations/Future Enhancements

USS Namenode High Availability

In this release, we have not implemented high availability for the USS Namenode. High Availability will be implemented in a future release.

USS across multiple HDFS versions

In this release, USS works with Pivotal HD 2.0.x HDFS. In a future release, we will implement support for other versions/implementations of HDFS like 1.x and MapR.

Security

The USS Namenode currently does not implement security. It re-uses file system permissioning/access control from the underlying file systems. In a future release, there may be a security layer in USS.

Catalog Service Integration

As described earlier in this document, currently, USS supports a postgres catalog. In a subsequent release, the catalog will reside in a central catalog service. The USS Namenode will communicate with the catalog service to resolve mount points.

Performance Testing/Benchmarking

For the alpha release, there has been no performance testing and benchmarking of USS. This will happen in a subsequent release and results will be published once available.

Limited Hive Support

In its alpha release, USS has limited support for hive as a higher level language. Hive is supported for the following scenario -

- The Hive warehouse directory is located on HDFS.
- External tables can be created with locations specified by USS URIs.

Support for more scenarios may be added in a subsequent release.

Troubleshooting/FAQs

- I am trying to access a file system through USS. However, I keep getting errors like `ipc.Client: Retrying connect to server: <USS Namenode Hostname>/<USS Namenode ip address>:<USS Namenode port>. Already tried 1 time(s).`'
  - Please check if the USS Namenode is running on the host and port specified in configuration against the 'uss.namenode.address' property.
- I am trying to access a file system through USS. However, I keep getting errors like `No FileSystem for scheme: uss`'
  - Make sure `fs.uss.impl` is set to `com.greenplum.hd.uss.USSProtocolTranslator` in `/etc/gphd/hadoop/conf/core-site.xml`
- Where do I specify the address of the USS Namenode and why?
  - The address of the USS Namenode can be specified on client and USS namenode hosts.

    Client Side:On the client host, the USS Namenode address can be specified by the property uss.fs.name in the file /etc/gphd/uss/conf/uss-client-site.xml. By default, on the client host, the USS Namenode address is set to localhost:16040. If users have set up the USS Namenode on a different host:port location, they can use this property to set the address of the USS Namenode.

    USS Namenode Side: The USS Namenode starts at localhost:16040 by default. If clients wish to change the port, they can specify the new port by setting the property uss.namenode.address on the USS Namenode host at the location /etc/gphd/uss/conf/uss-nn-site.xml.
- How do I configure USS as the default filesystem?
  - The cluster wide default file system can be over ridden to use USS by default. This implies that all files default to the file system URI scheme of `uss://` as opposed to `hdfs://`. This can be done by setting `fs.default.name` in `/etc/gphd/hadoop/conf/core-site.xml` to `uss://uss-namenode-host:uss-namenode-port` on all nodes in the cluster (datanodes, nodemanager, client hosts). The advantage of setting this in configuration is that you can use the USS mount-point only, to access a USS Path, as opposed to using the fully qualified USS URI. An example of this is:

```
$ hadoop fs -ls /

Found 1 items

drwxr-xr-x   - root supergroup          0 2012-10-15 21:20 /fs-test-mp ->
hdfs:/namenode:9000/user/testuser/fstest

$ hadoop fs -ls /fs-test-mp

Found 1 items

-rw-r--r--   3 root supergroup        199 2012-10-15 21:20
/user/testuser/fstest/README.txt
```

When using USS as the default FileSystem fpr mapreduce jobs, however, you need to set `fs.defaultFS` to `uss://<uss_namenode>:<uss_port>` in the `/etc/gphd/hadoop/conf/core-site.xml` on all nodemanager hosts. Also add `/etc/gphd/uss/conf/uss-client-site.xml` into `HADOOP_CLASSPATH` in `/etc/gphd/hadoop/confhadoop-env.sh` (or `/etc/default/hadoop`)

- Why do I see the exception below when I run a mapreduce job using USS Paths for input/output directories?

```
INFO mapred.JobClient: Task Id : attempt_201210151759_0001_m_000004_0,
Status : FAILED

java.lang.RuntimeException: java.lang.ClassNotFoundException:
com.greenplum.hd.uss.USSProtocolTranslator

    at
org.apache.hadoop.conf.Configuration.getClass(Configuration.java:867)

    at
org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:1380)

    at org.apache.hadoop.fs.FileSystem.access$200(FileSystem.java:66)

    at org.apache.hadoop.fs.FileSystem$Cache.get(FileSystem.java:1404)

    at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:254)

    at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:123)

    at org.apache.hadoop.mapred.Child$4.run(Child.java:254)

    at java.security.AccessController.doPrivileged(Native Method)

    at javax.security.auth.Subject.doAs(Subject.java:416)

    at
org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.j
ava:1121)

    at org.apache.hadoop.mapred.Child.main(Child.java:249)

Caused by: java.lang.ClassNotFoundException:
com.greenplum.hd.uss.USSProtocolTranslator

    at java.net.URLClassLoader$1.run(URLClassLoader.java:217)

    at java.security.AccessController.doPrivileged(Native Method)

    at java.net.URLClassLoader.findClass(URLClassLoader.java:205)

    at java.lang.ClassLoader.loadClass(ClassLoader.java:321)

    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:294)

    at java.lang.ClassLoader.loadClass(ClassLoader.java:266)

    at java.lang.Class.forName0(Native Method)

    at java.lang.Class.forName(Class.java:264)

    at
org.apache.hadoop.conf.Configuration.getClassByName(Configuration.java:820)

    at
```

```
org.apache.hadoop.conf.Configuration.getClass(Configuration.java:865)
    ... 10 more
```

- This is because the nodemanagers cannot find the USS library to resolve USS Paths at runtime. To fix this, please add the following lines to `/etc/gphd/hadoop/conf/hadoop-env.sh` and restart nodemanager on all nodes using `service hadoop-yarn-nodemanager restart`.

```
/etc/gphd/hadoop/conf/hadoop-env.sh
export HADOOP_CLASSPATH=/usr/lib/gphd/uss/uss-0.2.0.jar:/etc/gphd/uss/conf/
export HADOOP_USER_CLASSPATH_FIRST=true
```

- How do I list all the mount-points defined in the catalog?
  - USS provides an API to list all the mount-points that they point to. Users can list the mount-points by running a `hadoop fs -ls` query on the path '/' on any datanode or hadoop client node.

```
$ hadoop fs -ls uss://ussnn.mycompany.com:16040/
Found 1 items
drwxr-xr-x   - root supergroup          0 2012-10-15 21:20
uss://ussnn.mycompany.com:16040/fs-test-mp
drwxr-xr-x   - root supergroup          0 2012-10-15 21:20
uss://ussnn.mycompany.com:16040/wc-input
drwxr-xr-x   - root supergroup          0 2012-10-15 21:20
uss://ussnn.mycompany.com:16040/wc-output
```

Additionally, on the USS admin host, administrators can also use- uss list to list all the mount points and their targets.

```
$ uss list
click_data (1)   >
hdfs://namenode1.mycompany.com:8020/data/website/clicks
conv_data (2)    >
hdfs://namenode2.mycompany.com:8020/data/website/conversions
view_data (3)    >    file:///data/website/views
```

- What hadoop services are dependent on USS? In other words, which services need the USS library in their classpath at runtime?
  - In order to use USS for map-reduce, NodeManagers need to have the USS library in their classpath.

    To add the USS library to the classpath, which can be done by `source uss-env.sh`(or set `HADOOP_CLASSPATH` to the path of the USS jar, and set `HADOOP_USER_CLASSPATH_FIRST=true`). However, when you export these settings in the environment and run `service hadoop-yarn-nodemanager` `restart`, the `/etc/init.d/hadoop-yarn-nodemanager` script does a `sudo -u mapred hadoop-daemon.sh`. Because of the `sudo -u mapred`, a new shell gets spawned, and the environment variables do not propagate. As a result, even if you set these variables in your shell before starting the nodemanager, the nodemanager still does not have the USS jar in the classpath.

Instead, list the 2 settings here in `/etc/gphd/hadoop/conf/hadoop-env.sh` on all nodemanager nodes. This script is sourced in the bin/hadoop script, which is the last script executed while starting services (as a result, these environment settings will be present in the same shell).

- If a map-reduce job does file-system CRUD operations for exmple, it does a `FileSystem.open()` on a USS Path, then does any other service require USS?
  - In this case, the datanodes may need to have the `uss.jar` library available at runtime as well. Use the same technique described in the above answer to add the USS jar to the classpath of the datanode processes.

- If a USS configuration parameter value changes, what services need to be restarted for the new setting to take effect?
  - As of now, no USS setting is used by any of the hadoop services. `/etc/gphd/uss/conf/uss-client-site.xml` contains settings that are entirely used by the hadoop filesystem client. `/etc/gphd/uss/conf/uss-nn-site.xml` contains settings that are entirely used by the USS Namenode. Currently, we do not have any settings that may require us to restart hadoop services like namenode, secondary namenode, datanode, resourcemanager or nodemanager.

- I have configured USS to access HDFS running on a independent set of hosts, whose namenode is identified by the URI `hdfs://hdfs-nn.mydomain.com:8020/`. However, when I execute some Hadoop CLI commands using USS paths, I keep seeing Connection Refused errors. The errors seem to indicate that a connection attempt is being made to an HDFS Namenode on a different host/ip from my HDFS namenode host.
  - Please check the setting `uss.tmp.fs` in `/etc/gphd/uss/conf/uss-client-site.xml`. This setting is described in '"Configuring USS" on page 64 section in this document. Ensure that it is set to a location that is available to serve Hadoop Filesystem requests. USS uses it to store temporary data.

- When I run a pig script using USS, I see errors like `ClassNotFoundException: com.greenplum.hd.uss.USSProtocolTranslator`
  - Pig needs to include the USS library. You can achieve this by running your pig script as

```
$. pig -Dpig.additional.jars=/usr/lib/gphd/uss/uss-0.2.0.jar
<pig_script_using_uss_paths>
```

- When I run `uss` admin commands, I see a permission denied error.
  - Due to a known issue in USS, currently the {{uss admin }} commands must be run as sudo/root. This will be fixed in a future release.

- Why do I get an error `relation "mount_point" does not exist` when I try to add a mount-point using `uss admin -a`?
  - This error indicates that the table mount_point does not exist in the uss-catalog database. Please check that you have initialized the uss-catalog database using uss admin --initialize.

- Why do I get errors like `/var/log/gphd/uss-catalog/uss-catalog.log: Permission denied` when I try to run USS CLI commands?
  - Currently, the USS CLI commands have to be run either as root (sudo) or gpadmin. To make these commands accessible to other users, execute the following command as gpadmin/root -

    ```
    $ chmod -R 777 /var/log/gphd/uss-catalog
    ```

- USS processes (USS Namenode and USS CLI) don't use the Oracle jdk even though I have it installed and have `JAVA_HOME` set correctly.
  - In this version, all USS commands use the java executable pointed to by the `which java` command on your host. Please make sure that this points to the right version of java. This will be fixed in a subsequent release.

Terminology

- **USS URI**—A USS URI is a URI that USS can understand and resolve. It does not point to a 'real' location on a FileSystem, but contains an entry into a Catalog that can be used to lookup for the actual URI on an underlying 'Delegate FileSystem'. A USS URI has the form:

  ```
  uss://<USSNameNode Host>:<USSNameNode
  Port>/<Mount-Point>/<Sub-path under the mount-point>
  ```

- **Mount Point**—USS URI's are of the form: `uss://[Authority]/[Mount Point]/[Subdirectory]`. The mount point is an index in a 'mount-table' like Catalog system that USSNameNode can access. This Catalog could reside in Configuration or it could be part of the Centralized Catalog Service at a later stage. Given a USS URI, it is the responsibility of the USSNameNode to lookup the Catalog Service for the mount-point and return the 'Resolved URI' on a 'Delegate FileSystem'.
- **Catalog**—The Catalog is the metadata store that the USS Namenode uses to look up mount points and resolve USS URIs to their actual URIs on Delegate File Systems.

- **Resolved URI**—USS URI's are of the form 'uss://[Authority]/[Mount Point]/[Subdirectory]'. This URI does not point to a 'real' FileSystem, but is just an index into a Catalog. It is the responsibility of the USSNameNode to lookup the Catalog for the USS URI and resolve it to the correct location on an underlying 'Delegate FileSystem'. The 'real' URI returned after looking up the Catalog System is referred to in this document as a 'Resolved URI'. The 'Resolved URI' points to a real location on the 'Delegate FileSystem'
- **Delegate FileSystem**—USSProtocolTranslator provides an abstraction over a range of FileSystems in USS. All URIs in USS look like 'uss://[Authority]/[Mount Point]/[Subdirectory]'. Every time a URI lookup is desired, the call goes to USSProtocolTranslator which calls the USSNameNode to figure out the actual location of the URI, which resides on some underlying FileSystem (S3, HDFS, Isilon, Atmos, FTP, etc). The FileSystem call is then 'delegated' to this underlying FileSystem. This underlying FileSystem is referred to in this document as 'Delegate FileSystem'.
- **Client**—A client is the entity that sends requests to USS. A client could be a user using FsShell (Hadoop command line) to request access to a USS URI, a Java program that uses the Hadoop FileSystem API or a Map-Reduce program that uses the JobClient to setup a map-reduce job.

5. Security

Kerberos is a network authentication protocol that provides strong authentication for client/server applications using secret-key cryptography.

You must install and configure Kerberos to enable security in Pivotal HD 1.0.x.

HDFS, Mapreduce, Yarn, and Hive, and Pivotal HAWQ can be enabled for Kerberos.

Note: HAWQ does not currently work if HDFS is configured to use Kerberos.

This chapter contains the following:

Configuring Kerberos for HDFS and YARN (MapReduce)

At a minimum Kerberos provides protection against user and service spoofing attacks, and allows for enforcement of user HDFS access permissions. The installation is not difficult, but requires very specific instructions with many steps, and suffers from the same difficulties as any system requiring distributed configuration. Pivotal is working to automate the process to make it simple for users to enable/disable secure PHD clusters. Until then these instructions are intended to provide a step by step process for getting a cluster up and running in secure mode.

Note that after the initial HDFS/YARN configuration other services that need to be set-up to run on secure HDFS (for example, HBase) or that you want to also secure (for example, Zookeeper) need to configured.

Important: Save your command history, it will help in checking for errors when troubleshooting

Kerberos Set-up

Install the KDC

If you do not have a pre-existing KDC see "Installing the MIT Kerberos 5 KDC" on page 96.

Note: CentOS and RedHat use AES-256 as the default encryption strength. If you want to use AES-256 you need to install the JCE security policy file (described below) on all cluster hosts. If not, disable this encryption type in the KDC configuration. To disable AES-256 on an MIT kerberos 5 KDC remove `aes256-cts:normal` from the `supported_enctypes` parameter in `kdc.conf`.

Integrating Cluster Security with an Organizational KDC

If your organization runs Active Directory or other Kerberos KDC it is not recommended this be used for cluster security. Instead install an MIT Kerberos KDC and realm for the cluster(s) and create all the service principals in this realm as per the instructions below. This KDC will be minimally used for service principals whilst Active Directory (or your organizations's MIT KDC) will be used for cluster users. Next configure one-way cross-realm trust from this realm to the Active Directory or corporate KDC realm.

Important: This is strongly recommended as a large PHD cluster requires large numbers of service principals be created by the IT manager for your organizations' Active Directory or organizational MIT KDC. For example a 100 node PHD cluster requires 200+ service principals. In addition when a large cluster starts up it may impact the performance of your organizations' IT systems as all the service principals make requests of the AD or MIT Kerberos KDC at once.

Install Kerberos Workstation and Libraries on Cluster Hosts

If you are using MIT krb5 run the following command:

```
yum install krb5-libs krb5-workstation
```

Distribute the Kerberos Client Configuration File to all Cluster Hosts

If you are using Kerberos 5 MIT this is `/etc/krb5.conf`. This file must exist on all cluster hosts. For PHD you can use `massh` to push the files, and then to copy them to the proper place.

Create the Principals

These instructions are for MIT Kerberos 5, command syntax for other Kerberos versions may be different.

Principals (Kerberos users) are of the form: *`name`*`/role@REALM`. For our purposes the name will be a PHD service name (for example: `hdfs`) and the role will be a DNS resolvable hostname (one you can use to connect to the host in question).

Important:

- The hostnames used MUST be resolvable to an address on all the cluster hosts.
- The names of the principals seem to matter as some processes may throw exceptions if you change them. Hence it is safest to use these principal names if possible.

For the HDFS services you will need to create an `hdfs/`*`hostname`* principal for each host running an HDFS service (namenode, secondary namenode, datanode). For YARN services you will need to create a `yarn/`*`hostname`* principal for each host running a YARN service (resource manager, node manager). For MapReduce services you need to create a principal `mapred/`*`hostname`* for the Job History Server.

To create the required secure HD principals (using krb5 command syntax):

- For each cluster host (except client-only hosts) run:
  ```
  addprinc -randkey HTTP/hostname@REALM
  ```
- HDFS (namenode, datanodes), for each HDFS service host, run:
  ```
  addprinc -randkey hdfs/hostname@REALM
  ```

- YARN (resource manager, node managers), for each YARN service host, run:
  ```
  addprinc -randkey yarn/hostname@REALM
  ```
- MAPRED (job history server): for each JHS service host, run:
  ```
  addprinc -randkey  mapred/hostname@REALM
  ```

Important: If you have 1000 cluster hosts running HDFS and YARN you will need 2000 HDFS and YARN principals, and need to distribute their keytab files. It is recommended you use a cluster-local KDC for this purpose and configure cross-realm trust to your organizational Active Directory or other Kerberos KDC.

Create the Keytab Files

Important: You MUST use `kadmin.local` (or the equivalent in your KDC) for this step on the KDC as kadmin does not support `-norandkey`.

Important: You can put the keytab files anywhere during this step, in this document we are creating a `directory /etc/security/keytab/` and using that on cluster hosts, and so for consistency are placing them in a similarly named directory on the KDC. If the node you are on already has files in `/etc/security/keytab/` it may be best to create a separate, empty, directory for this step.

Each service's keytab file for a given host will contain the service principal for that host and the HTTP principal for that host.

HDFS key tabs

For each host having an HDFS process (resource manager or node manager) run:

```
kadmin.local:  ktadd -norandkey -k
/etc/security/keytab/hdfs-hostid.service.keytab
hdfs/hostname@REALM  HTTP/hostname@REALM
```

Where `hostid` is just a short name for the host, for example, `vm1`, `vm2`, etc. This is to differentiate the files by host. You can use the *`hostname`* if desired.

For example for a 3 node cluster (one namenode, 2 datanodes):

```
kadmin.local: ktadd -norandkey -k
/etc/security/keytab/hdfs-vm2.service.keytab
hdfs/centos62-2.localdomain@BIGDATA.COM
HTTP/centos62-2.localdomain@BIGDATA.COM

kadmin.local: ktadd -norandkey -k
/etc/security/keytab/hdfs-vm3.service.keytab
hdfs/centos62-3.localdomain@BIGDATA.COM
HTTP/centos62-3.localdomain@BIGDATA.COM

kadmin.local: ktadd -norandkey -k
/etc/security/keytab/hdfs-vm4.service.keytab
hdfs/centos62-4.localdomain@BIGDATA.COM
HTTP/centos62-4.localdomain@BIGDATA.COM
```

YARN keytabs

For each host having a YARN process (resource manager or node manager) run:

```
kadmin.local:  ktadd -norandkey -k
/etc/security/keytab/yarn-hostid.service.keytab
yarn/hostname@REALM   HTTP/hostname@REALM
```

For example, for a 3 node cluster (one node resource manager; 2 node managers):

```
kadmin.local: ktadd -norandkey -k
/etc/security/keytab/yarn-vm2.service.keytab
yarn/centos62-2.localdomain@BIGDATA.COM
HTTP/centos62-2.localdomain@BIGDATA.COM
kadmin.local: ktadd -norandkey -k
/etc/security/keytab/yarn-vm3.service.keytab
yarn/centos62-3.localdomain@BIGDATA.COM
HTTP/centos62-3.localdomain@BIGDATA.COM
kadmin.local: ktadd -norandkey -k
/etc/security/keytab/yarn-vm4.service.keytab
yarn/centos62-4.localdomain@BIGDATA.COM
HTTP/centos62-4.localdomain@BIGDATA.COM
```

MAPRED keytabs

For each host having a MapReduce job history server run:

```
kadmin.local:  ktadd -norandkey -k
/etc/security/keytab/mapred-hostid.service.keytab
mapred/hostname@REALM  HTTP/hostname@REALM
```

For example:

```
kadmin.local: ktadd -norandkey -k
/etc/security/keytab/mapred-vm2.service.keytab
mapred/centos62-2.localdomain@BIGDATA.COM
HTTP/centos62-2.localdomain@BIGDATA.COM
```

Distribute the Keytab Files

1. On each cluster node create the directory for the keytab files; in this document we are using `/etc/security/keytab`.
2. Move all the keytab files for a given host to the keytab directory on that host. For example: `hdfs-vm2.service.keytab`, `yarn-vm2.service.keytab`, and `mapred-vm2.service.keytab` go to host `vm2`)
3. On each host:
   a. Change the permissions on all keytabs to read-write by owner only:
      `chmod 400  *.keytab`
   b. Change the group on all keytab files to hadoop:
      `chgrp hadoop *`
   c. Change the owner of each keytab to the relevant principal name.
      For example, for `yarn-vm2.service.keytab` run:
      `chown yarn yarn-vm2.service.keytab`
   d. Create links to the files of the form `principalname.service.keytab`.
      For example, for `yarn-vm2.service.keytab` run:
      `ln -s yarn-vm2.service.keytab yarn.service.keytab`

Important: The last step above allows you to maintain clear identification of each keytab file while also allowing you to have common site xml files across cluster hosts.

Below is an example keytab directory for a cluster control node (namenode, resource manager, JHS):

```
lrwxrwxrwx 1 root      root    23 Jun 10 23:50
hdfs.service.keytab -> hdfs-vm2.service.keytab
-rw------- 1 hdfs      hadoop 954 Jun 10 23:44
hdfs-vm2.service.keytab
lrwxrwxrwx 1 root      root    25 Jun 10 23:51
mapred.service.keytab -> mapred-vm2.service.keytab
-rw------- 1 mapred    hadoop 966 Jun 10 23:44
mapred-vm2.service.keytab
lrwxrwxrwx 1 root      root    23 Jun 10 23:51
yarn.service.keytab -> yarn-vm2.service.keytab
-rw------- 1 yarn      hadoop 954 Jun 10 23:44
yarn-vm2.service.keytab
```

Below is an example keytab directory for a cluster node (datanode, node manager):

```
lrwxrwxrwx 1 root root   23 Jun 11 01:58 hdfs.service.keytab
-> hdfs-vm3.service.keytab
-rw------- 1 hdfs hadoop 954 Jun 10 23:45
hdfs-vm3.service.keytab
lrwxrwxrwx 1 root root   23 Jun 11 01:58 yarn.service.keytab
-> yarn-vm3.service.keytab
-rw------- 1 yarn hadoop 954 Jun 10 23:45
yarn-vm3.service.keytab
```

Install Java Support Items

Install JCE on all Cluster Hosts

Important: This step is only needed if you are using AES-256

Note: These files will already exist in your environment and appear to be the same, but are the limited strength encryption files, you must replace them with the unlimited strength files to use AES-256

1. Download and unzip the JCE file for your JDK version (Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7 for JDK 7 and Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6 for JDK6).
2. Place the `local_policy.jar` and `US_export_policy.jar` files in the `/usr/java/default/jre/lib/security/` directory on all cluster hosts.

Check JSVC on all Datanodes

JSVC allows a Java process to start as root and then switch to a less privileged user, and is required for the datanode process to start in secure mode. Your distribution comes with a pre-built JSVC, you need to verify it can find a JVM as follows:

1. Run:

```
/usr/libexec/bigtop-utils/jsvc  -help
```

2. Look under the printed `-jvm` item in the output and you should see something like:

   ```
   use a specific Java Virtual Machine. Available JVMs:
   'server'
   ```

   If you do not see the `server` line this jsvc will not work for your platform so try:

3. Install JSVC using yum and run the check again; if it fails try **4.)** below:
4. Build from source and install manually (see "Building and Installing JSVC" on page 95).

If you have datanode start-up problems and no other errors are obvious it might be a JSVC problem and you may need to do item **4.)** above. JSVC is very precise about platform and JDK matching, so use the instructions in "Building and Installing JSVC" on page 95 for your system OS and JDK.

Container and Script Modifications

Configure the Linux Container

1. Edit the `/usr/lib/gphd/hadoop-yarn/etc/hadoop/container-executor.cfg` to be:

   ```
   # NOTE: these next two should be set to the same values they
   have in yarn-site.xml
   yarn.nodemanager.local-dirs=/data/1/yarn/nm-local-dir
   yarn.nodemanager.log-dirs=/data/1/yarn/userlogs
   # configured value of
   yarn.nodemanager.linux-container-executor.group
   yarn.nodemanager.linux-container-executor.group=yarn
   # comma separated list of users who can not run applications
   banned.users=hdfs,yarn,mapred,bin
   # Prevent other super-users
   min.user.id=500
   ```

Note: The `min.user.id` varies by Linux dist; for CentOS it is 500, RedHat is 1000.

2. Check the permissions on `/usr/lib/gphd/hadoop-yarn/bin/container-executor`. They should look like this:

   ```
   ---Sr-s--- 1 root yarn   364 Jun 11 00:08 container-executor
   ```

   If they do not, then set the owner, group and permissions as:

   ```
   chown root:yarn container-executor
   chmod 050 container-executor
   chmod u+s container-executor
   chmod g+s container-executor
   ```

3. Check the permissions on `/usr/lib/gphd/hadoop-yarn/etc/hadoop/container-executor.cfg`. They should look like this:

   ```
   -rw-r--r-- 1 root root 363 Jul  4 00:29
   container-executor.cfg
   ```

   If they do not, then set the owner, group and permissions as:

   ```
   chown root:root container-executor.cfg
   chmod 644 container-executor.cfg
   ```

Edit the Environment on the Datanodes

Important:

- At this point you should STOP the cluster if it is running
- You only need to do the steps below on the datanodes

1. Uncomment the lines at the end of `/etc/default/hadoop-hdfs-datanode`:

   ```
   # secure operation stuff
   export HADOOP_SECURE_DN_USER=hdfs
   export HADOOP_SECURE_DN_LOG_DIR=${HADOOP_LOG_DIR}/hdfs
   export HADOOP_PID_DIR=/var/run/gphd/hadoop-hdfs/
   export HADOOP_SECURE_DN_PID_DIR=${HADOOP_PID_DIR}
   ```

2. Set the JSVC variable:

   If you are using the included jsvc the `JSVC_HOME` variable in `/etc/default/hadoop` should already be properly set:

   ```
   export JSVC_HOME=/usr/libexec/bigtop-utils
   ```

   If you built or manually installed JSVC your `JSVC_HOME` will be `/usr/bin` and you must set it appropriately by modifying `/etc/default/hadoop` and setting the proper `JSVC_HOME`:

   ```
   export JSVC_HOME=/usr/bin
   ```

   Important: Make sure `JSVC_HOME` points to the correct jsvc binary

WARNING: As long as `HADOOP_SECURE_DN_USER` is set the datanode will try and start in secure mode.

Site XML Changes

Edit the Site XML

Finally you need to edit the site XML to turn on secure mode. Before proceeding it is good to understand who needs to talk to whom. By "talk" we mean use authenticated kerberos to initiate establishment of a communication channel. Doing this requires that you know your own principal to identify yourself and know the principal of the service you want to talk to. To be able to use its principal, a service needs to be able to login to Kerberos without a password using a keytab file.

- Each service needs to know its own principal name, of course
- Each running service on a node needs a service/host specific keytab file to start up
- Each datanode needs to talk to the namenode
- Each node manager needs to talk to the resource manager and the job history server
- Each client/gateway node needs to talk to the namenode, resource manager, and job history server

Remembering this helps when setting up and troubleshooting the site xml files.

Important:

- Redundant keytab files on some hosts do no harm and it makes management easier to have constant files. Remember though that the hostname MUST be correct for each entry.
- Before making changes, backup the current site xml files so that you can return to non-secure operation if needed.

Much of the changes can be consistent throughout the cluster site XML, but since datanode and node manager principals are hostname dependent (or more correctly the role for the yarn principal is set to the hostname), the `yarn-site.xml` for datanode and node manager principals will differ across the cluster.

1. Edit the `/usr/lib/gphd/hadoop/etc/hadoop/core-site.xml` as follows:

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

<property>
   <name>hadoop.security.authorization</name>
   <value>true</value>
</property>
```

2. Edit the `/usr/lib/gphd/hadoop/etc/hadoop/hdfs-site.xml` as follows:

```
<!-- WARNING: do not create duplicate entries: check for
existing entries and modify if they exist! -->
```

```
<property>
  <name>dfs.block.access.token.enable</name>
  <value>true</value>
</property>

<!-- namenode secure configuration info -->

<property>
  <name>dfs.namenode.keytab.file</name>
  <value>/etc/security/keytab/hdfs.service.keytab</value>
</property>

<property>
  <name>dfs.namenode.kerberos.principal</name>
  <value>hdfs/namenode-hostname@REALM</value>
</property>

<property>
  <name>dfs.namenode.kerberos.http.principal</name>
  <value>HTTP/namenode-hostname@REALM</value>
</property>

<property>

<name>dfs.namenode.kerberos.internal.spnego.principal</name>
  <value>HTTP/namenode-hostname@REALM</value>
</property>

<!-- ?(optional) secondary namenode secure configuration
info -->

<property>
  <name>dfs.secondary.namenode.keytab.file</name>
  <value>/etc/security/keytab/hdfs.service.keytab</value>
</property>

<property>
  <name>dfs.secondary.namenode.kerberos.principal</name>
  <value>hdfs/secondary-namenode-hostname@REALM</value>
</property>

<property>
```

```
<name>dfs.secondary.namenode.kerberos.http.principal</name>
  <value>HTTP/secondary-namenode-hostname@REALM</value>
</property>

<property>

<name>dfs.secondary.namenode.kerberos.internal.spnego.princi
pal</name>
  <value>HTTP/secondary-namenode-hostname@REALM</value>
</property>

<!-- datanode secure configuration info -->

<property>
  <name>dfs.datanode.data.dir.perm</name>
  <value>700</value>
</property>

<!-- these ports must be set < 1024 for secure operation -->
<!-- conversely they must be set back to > 1024 for
non-secure operation -->
<property>
  <name>dfs.datanode.address</name>
  <value>0.0.0.0:1004</value>
</property>

<property>
  <name>dfs.datanode.http.address</name>
  <value>0.0.0.0:1006</value>
</property>

<!-- remember the principal for the datanode is the
principal this hdfs-site.xml file is on -->

<!-- these (next three) need only be set on datanodes -->

<property>
  <name>dfs.datanode.kerberos.principal</name>
  <value>hdfs/this-datanodes-hostname@REALM</value>
</property>

<property>
  <name>dfs.datanode.kerberos.http.principal</name>
  <value>HTTP/this-datanodes-hostname@REALM</value>
</property>

<property>
  <name>dfs.datanode.keytab.file</name>
  <value>/etc/security/keytab/hdfs.service.keytab</value>
</property>

<!-- OPTIONAL - set these to enable secure WebHDSF -->

<!-- on all HDFS cluster nodes -->
<!--(namenode, secondary namenode, datanode's) -->

<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>

<property>
  <name>dfs.web.authentication.kerberos.principal</name>
  <value>HTTP/this-datanodes-hostname@REALM</value>
</property>

<!-- since we included the HTTP principal all -->
<!-- keytabs we can use it here -->

<property>
  <name>dfs.web.authentication.kerberos.keytab</name>
  <value>/etc/security/keytab/hdfs.service.keytab</value>
</property>
```

3. Edit the `/usr/lib/gphd/hadoop/etc/hadoop/yarn-site.xml` as follows:

```
<!-- resource manager secure configuration info -->

<property>
  <name>yarn.resourcemanager.principal</name>
  <value>yarn/resourcemgr-hostname@REALM</value>
</property>

<property>
  <name>yarn.resourcemanager.keytab</name>
  <value>/etc/security/keytab/yarn.service.keytab</value>
</property>

<!-- remember the principal for the node manager is the
principal for the host this yarn-site.xml file is on -->

<!-- these (next four) need only be set on node manager nodes
-->

<property>
  <name>yarn.nodemanager.principal</name>
  <value>yarn/this-nodemgrs-hostname@REALM</value>
</property>

<property>
  <name>yarn.nodemanager.keytab</name>
  <value>/etc/security/keytab/yarn.service.keytab</value>
</property>

<property>
  <name>yarn.nodemanager.container-executor.class</name>

<value>org.apache.hadoop.yarn.server.nodemanager.LinuxContai
nerExecutor</value>
</property>

<property>

<name>yarn.nodemanager.linux-container-executor.group</name>
  <value>yarn</value>
</property>

edit /usr/lib/gphd/hadoop/etc/hadoop/mapred-site.xml:

<!-- job history server secure configuration info -->

<property>
  <name>mapreduce.jobhistory.keytab</name>
  <value>/etc/security/keytab/mapred.service.keytab</value>
</property>

<property>
  <name>mapreduce.jobhistory.principal</name>
  <value>mapred/jobhistoryserver-hostname@REALM</value>
</property>
```

Complete the HDFS/YARN Secure Configuration

1. Start the cluster:

   ```
   icm_client start
   ```

2. Check that all the processes start up. If not, refer to "Troubleshooting" on page 108.

   - **Control Processes**: namenode, resourcemanager, historyserver should all be running.
   - **Cluster Worker Processes**: datanode and namenode should be running.

     Note: Until you configure HBase security configuration, HBase will not start up on a secure cluster.

3. Create a principal for a standard user (user must exist as a Linux user on all cluster hosts):

   ```
   kadmin: addprinc testuser
   ```

   Set the password when prompted.

4. Login as that user on a client box (or any cluster box if you do not have specific client purposed systems).

5. Get your kerberos TGT by running kinit and entering the password: kinit testuser.

6. Test simple HDFS file list and directory create:

   ```
   hadoop fs -ls
   hadoop fs -mkdir testdir
   ```

If these do not work refer to "Troubleshooting" on page 108.

7. Optional. Set the sticky bit on the `/tmp` directory (prevents non-super users from moving or deleting other users' files in `/tmp`)

   a. Login as `gpadmin` on any HDFS service node (namenode, datanode)

   b. Execute the following:

   ```
   sudo -u hdfs kinit -k -t
   /etc/security/keytab/hdfs.service.keytab
   hdfs/this-hostname@REALM
   ```

   c. Execute the following:

   ```
   sudo -u hdfs hadoop fs -chmod 1777 /tmp
   ```

   d. Run a simple MapReduce job such as the pi example:

   ```
   hadoop jar
   /usr/lib/gphd/hadoop-mapreduce/hadoop-mapreduce-examples-
   2.0.2-alpha-gphd-2.0.1.0.jar pi 10 100
   ```

If this all works then you are ready to configure other services, if not refer to "Troubleshooting" on page 108.

Turning Secure Mode Off

To turn off secure mode:

1. Stop the cluster:

   ```
   icm_client stop
   ```

2. Comment out `HADOOP_SECURE_DN_USER` in `hadoop-env.sh` and `/etc/init.d/hadoop-hdfs-datanode` on all datanodes

3. Either:

   a. If you made backups as suggested above:
      Restore the original site xml files.

   or:

   b. If you do not have backups, then edit the site xml, as follows:

   - Set Linux container executable to `org.apache.hadoop.yarn.server.nodemanager.DefaultContainerExecutor` on all datanodes.
   - Set `dfs.block.access.token.enable` to `false` on all datanodes
   - Return the datanode ports modified above so they are > 1024 again.
   - Set `hadoop.security.authentication` to `simple` and `hadoop.security.authorization` to `false` in `core-site.xml` on all cluster nodes.
   - Start the cluster: `icm_client start`

Building and Installing JSVC

In order for the datanodes to start as root to get secure ports and then switch back to the hdfs user, JSVC must be installed. If the packaged jsvc binary is not working we recommend building jscv from source for your platform.

You only need to execute the make on one node, then the binary can be distributed to the others (assuming all systems are the same basic image):

1. Install gcc and make (you can remove them after this process if desired).

   ```
   yum install gcc make
   ```

2. Download the Apache commons daemon; for example.
   `commons-daemon-1.0.15-src.zip` was tested.

   The demon is available here:
   http://commons.apache.org/proper/commons-daemon/download_daemon.cgi

3. `scp` it to one of your datanode cluster systems.

4. Uncompress it.

5. Change to the install directory:

   ```
   cd commons-daemon-1.0.15-src/src/native/unix
   ```

6. If you are on a 64 bit machine and using a 64 bit JVM perform these exports before continuing to configure/make:

   ```
   export CFLAGS=-m64
   export LDFLAGS=-m64
   ```

7. Configure and make it:

   ```
   ./configure --with-java=/usr/java/default
   make
   ```

8. Manually install it to the following location:

   ```
   mv ./jsvc  /usr/bin/jsvc
   ```

9. Check that the correct jsvc is found by doing running:

   ```
   which jsvc
   ```

   The correct output is:

   ```
   /usr/bin/jsvc
   ```

10. Run:

    ```
    jsvc  -help
    ```

    Look under the printed `-jvm` item in the output and you should see something like:

    ```
    use a specific Java Virtual Machine. Available JVMs:
    'server'
    ```

    If the line under `Available JVMs` (where `server` appears above) is blank there is a problem as it cannot find the JVM; check that the JDK is installed properly in `/usr/java/default`.

Installing the MIT Kerberos 5 KDC

This section outlines a simple krb5 KDC set-up, mainly for test and developer purposes. These instructions were largely derived from "*Kerberos: The Definitive Guide*", James Garman, O'Reilly, pages 53-62.

1. Install the Kerberos packages `krb5-libs`, `krb5-workstation`, and `krb5-server` on the KDC host.
2. Define your REALM in `/etc/krb5.conf`.
   - For testing you can use the `EXAMPLE.COM REALM`.
   - Set the `kdc` and `admin_server` variables to the resolvable hostname of the KDC host.
   - Set the `default_domain` to your REALM.

In the following example, REALM was changed to `BIGDATA.COM` and the KDC host is `centos62-1.localdomain`:

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
```

```
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 default_realm = BIGDATA.COM
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true

[realms]
 BIGDATA.COM = {
  kdc = centos62-1.localdomain:88
  admin_server = centos62-1.localdomain:749
  default_domain = BIGDATA.COM
 }

[domain_realm]
 .bigdata.com = BIGDATA.COM
 bigdata.com = BIGDATA.COM
```

3. Set up `/var/kerberos/krb5kdc/kdc.conf`.
   - If you want to use AES-256, uncomment the `master_key_type` line.
   - If you do not want to use AES-256, remove it from the `supported_enctypes` line
   - Add a `key_stash_file` entry: `/var/kerberos/krb5kdc/.k5.REALM`.
   - Add the `kadmind_port` entry: `kadmind_port = 749`.

   Important: The stash file lets the KDC server start up for root without a password being entered.

   The result (using AES-256) for the above REALM is:

```
[kdcdefaults]
 kdc_ports = 88
 kdc_tcp_ports = 88

[realms]
 BIGDATA.COM = {
  master_key_type = aes256-cts
  acl_file = /var/kerberos/krb5kdc/kadm5.acl
  dict_file = /usr/share/dict/words
  admin_keytab = /var/kerberos/krb5kdc/kadm5.keytab
  key_stash_file = /var/kerberos/krb5kdc/.k5.BIGDATA.COM
  kadmind_port = 749
  supported_enctypes = aes256-cts:normal
aes128-cts:normal des3-hmac-sha1:normal
arcfour-hmac:normal des-hmac-sha1:normal
des-cbc-md5:normal des-cbc-crc:normal
 }
```

4. Create the KDC master password:

   `run: kdb5_util create -s`

   DO NOT forget your password as this is the root KDC password.

5. Add an administrator account as `username/admin@REALM`:

   `start kadmin.local`

   `kadmin.local: addprinc username/admin@REALM`

   Important: The KDC does not need to be running to add a principal.

6. Start the KDC by running:

   `service krb5kdc start`

   You should get an `[OK]` indication if it started without error.

7. Edit `/var/kerberos/krb5kdc/kadm5.acl` and change the admin permissions username from `*` to your admin.

   You can add other admins with specific permissions if you want (`man kadmind`)

   This is a sample ACL file:

   ```
   joeit/admin@BIGDATA.COM       *
   ```

8. Use kadmin.local on the KDC to enable the administrator(s) remote access:

   ```
   kadmind.local: ktadd -k /var/kerberos/krb5kdc/kadm5.keytab
   kadmin/admin kadmin/changepw
   ```

   Important: `kadmin.local` is a KDC host only version of `kadmin` that can do things remote kadmin cannot (such as use the `-norandkey` option in `ktadd`)

9. Start kadmind:

   `service kadmin start`

   The KDC should now be done and ready to use, but you need to set up your clients first.

10. Install `krb5-libs` and `krb5-workstation` on all cluster hosts, including any client/gateway hosts.

11. Push your KDC `/etc/krb5.conf` to all workstation hosts.

12. Do a simple test, as follows:

    a. Login as the admin you created: `kinit username/admin`

    b. Run `kadmin` and make sure you can login

       If you get the message `kinit: Cannot contact any KDC for realm 'REALM' while getting initial credentials`, then the KDC is not running or the KDC host information in `/etc/kdc.conf` is incorrect.

You now should have a KDC that is functional for PHD secure cluster operations.

Zookeeper Secure Configuration

Zookeeper secure configuration for server is recommended for HBase.

Important: STOP cluster services before performing this configuration.

Note: Secure HBase on Pivotal HD is not yet certified for operations with a secure Zookeeper.

Zookeeper Servers

Create the Zookeeper Principals:

1. Create a principal for each Zookeeper Quorum Server host:

   ```
   kadmin: addprinc -randkey zookeeper/hostname@REALM
   ```

Create the Zookeeper Keytab Files

1. For each Zookeeper server host:

   ```
   ktadd -norandkey -k
   /etc/security/keytab/zookeeper-hostid.service.keytab
   zookeeper/hostname@REALM
   ```

Distribute the Zookeeper Keytab Files

1. For each Zookeeper server host:

   Move the appropriate keytab file for each host to that hosts' `/etc/security/keytab directory`, then run the following:

   ```
   chgrp hadoop zookeeper-hostid.service.keytab
   chown zookepper zookeeper-hostid.service.keytab
   chmod 400 zookeeper-hostid.service.keytab
   ln -s zookeeper-hostid.service.keytab
   zookeeper.service.keytab
   ```

Edit the Zookeeper Configuration

1. Add the following lines to `/etc/gphd/zookeeper/conf/zoo.cfg`:

   ```
   authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenti
   cationProvider
   jaasLoginRenew=3600000
   ```

2. Create a file in `/etc/gphd/zookeeper/conf/jaas.conf` and add to it:

   ```
   Server {
      com.sun.security.auth.module.Krb5LoginModule required
      useKeyTab=true

      keyTab="/etc/security/keytab/zookeeper-hostid.service.key
      tab"
   ```

```
      storeKey=true
      useTicketCache=false
      principal="zookeeper/hostname@REALM";
   };
```

3. Add the following line to `/etc/gphd/zookeeper/conf/java.env` (create the file if it does not exist):

   ```
   export
   JVMFLAGS="-Djava.security.auth.login.config=/etc/gphd/zookee
   per/conf/jaas.conf"
   ```

Verify the Zookeeper Configuration

1. Start up the cluster and connect using a client.

   Note: You do not need to set up clients to use Kerberos but if you want this functionality see "Zookeeper Clients" below:

2. Connect as: `zookeeper-client -server hostname:port`

   Note: the port is defined in `/etc/gphd/zookeeper/conf/zoo.cfg`, and is typically 2181.

3. Create a protected znode:

   ```
   create /testznode testznodedata sasl:zkcli@REALM:cdwra
   ```

4. Verify the znode:

   ```
   getAcl /testznode
   ```

   You should see something like this:

   ```
   'sasl,'zkcli@{{BIGDATA.COM%7D%7D
   : cdrwa
   ```

Zookeeper Clients

Optional

1. Add a principal for the client on the client host:

   ```
   kadmin.local: addprinc -randkey zclient/hostname@REALM
   ```

2. Add the keytab:

   ```
   kadmin.local: ktadd -norandkey -k
   /etc/security/keytab/zclient-hostid.client.keytab
   zclient/hostname@REALM
   ```

3. Move the file to the `/etc/security/keytab` directory on the host and change the owner and group appropriately so that only users of the client can access the file:

   ```
   chmod 400 /etc/security/keytab/zclient-hostid.client.keytab
   ```

4. Create a link:

   ```
   ln -s zclient-hostid.client.keytab zclient.client.keytab
   ```

5. Add the following to the file `/etc/gphd/zookeeper/conf/jaas.conf` (creating the file if required):

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="/etc/security/keytab/zclient.client.keytab"
  storeKey=true
  useTicketCache=false
  principal="zclient/hostname@REALM";
};
```

If you get a failure message indicating a name lookup failure that indicates you should add a name service setting, add or edit the following line to `/etc/gphd/zookeeper/conf/java.env` (create the file if it does not exist) to be:

```
export
JVMFLAGS="-Djava.security.auth.login.config=/etc/gphd/zookee
per/conf/jaas.conf
-Dsun.net.spi.nameservice.provider.1=dns,sun"
```

You cannot do this on a server node as the `-Dsun.net.spi.nameservice.provider.1=dns,sun` line may cause the server to fail to start.

You should now be able to establish a secure session with zookeeper-client. Test this by staring zookeeper-client and insuring no errors occur while connecting.

You may have issues with addressing or be forced to use the actual server IP address with the `-server` option for zookeeper-client to handle incompatibilities between the settings needed to make the Kerberos lookups work (`-Dsun.net.spi.nameservice.provider.1=dns,sun`) and what makes the Java host resolution work. This problem also may be encountered in trying to set up HBase to communicate with a secure Zookeeper, where it is more difficult to resolve.

HBase Secure Configuration

If you are running secure HBase you should also also run a secure Zookeeper (see "Zookeeper Secure Configuration"). You can, however, set the HBase master and region servers up to use Kerberos and test that they start without a secure Zookeeper. This section covers the basics of how to get HBase up and running in secure mode; for further information see the HBase documentation (`http://hbase.apache.org/book/security.html`).

Note: Secure HBase on Pivotal HD is not yet certified for operations with a secure Zookeeper.

HBase Master and Regionservers

Create the HBase Principals

1. For the HBase master and each region server host run:

```
kadmin.local: addprinc -randkey hbase/hostname@REALM
```

Where `hostname` refers to the service principal (master, regionserver) host.

Create the HBase Keytab files

1. For the HBase master and each region server host run:

   ```
   kadmin.local: ktadd -norandkey -k
   /etc/security/keytab/hbase-hostid.service.keytab
   hbase/hostname@REALM
   ```

Distribute the HBase Keytab Files

1. For each host:

   Move the appropriate keytab file to that hosts' `/etc/security/keytab` directory, then run the following:

   ```
   chown hbase:hadoop hbase-hostid.service.keytab
   chmod 400 hbase-hostid.service.keytab
   ln -s hbase-hostid.service.keytab hbase.service.keytab
   ```

Edit the HBase Site XML

1. For each master and region server host add to `/etc/gphd/hbase/conf/hbase-site.xml`:

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

   <property>
      <name>hbase.security.authorization</name>
      <value>true</value>
   </property>

   <property>
      <name>hbase.coprocessor.region.classes</name>

      <value>org.apache.hadoop.hbase.security.token.TokenProvid
      er</value>
   </property>

   <!-- HBase secure region server configuration -->
   <property>
      <name>hbase.regionserver.kerberos.principal</name>
      <value>hbase/regionserver-hostname@REALM</value>
   </property>

   <property>
   ```

```
   <name>hbase.regionserver.keytab.file</name>

   <value>/etc/security/keytab/hbase.service.keytab</value>
</property>

<!-- HBase secure master configuration -->
<property>
   <name>hbase.master.kerberos.principal</name>
   <value>hbase/master-hostname@REALM</value>
</property>

<property>
   <name>hbase.master.keytab.file</name>

   <value>/etc/security/keytab/hbase.service.keytab</value>
</property>
```

Test HBase Start-Up

You can now test HBase start-up. Start the cluster services and check that the HBase Master and Regionservers start properly. If they do not look at the `.log` file in the `/var/log/gphd/hbase/` directory for hints as to why. Make sure HDFS came up properly. As you fix issues you can run `service hbase-master start` or `service hbase-regionserver start` to check that the issue is resolved.

HBase Clients

1. Add the following to the `hbase-site.xml` file on every client host:

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

Enable Encrypted Communication

Optional

If you are running secure HBase you can enable encryption from clients to server by adding the following to `hbase-site.xml` on all clients:

```
<property>
   <name>hbase.rpc.protection</name>
   <value>privacy</value>
</property>
```

This can also be set on a per-connection basis. Set it in the Configuration supplied to HTable:

```
Configuration conf = HBaseConfiguration.create();
conf.set("hbase.rpc.protection", "privacy");
```

```
HTable table = new HTable(conf, tablename);
```

The Apache HBase documentation indicates to expect a ~10% performance penalty when encryption is enabled.

Access Control

The version of HBase distributed with PHD supports access control. See the HBase documentation (http://hbase.apache.org/book/hbase.accesscontrol.configuration.html) for instructions on configuring access controls.

REST Gateway

You can set up the REST Gateway to use Kerberos to authenticate itself as a principal to HBase. Note that all client access will use the REST Gateway's credentials set below, and have this user's privileges.

For every REST Gateway add the following to `hbase-site.xml` file:

```
<property>
   <name>hbase.rest.keytab.file</name>
   <value>path-to-rest-users-keytab</value>
</property>


<property>
   <name>hbase.rest.kerberos.principal</name>
   <value>rest-users-principal-name</value>
</property>
```

You must also give the REST principal access privileges. Do this by adding the rest-principal-name to the acl table in HBase. Adding the permissions below are sufficient according to HBase documentation:

```
grant 'rest-principal-name', 'RWCA'
```

Thrift Client Configuration

See the HBase documentation (http://hbase.apache.org/book/security.html) for instructions on configuring Thrift clients.

HBase with Secure Zookeeper Configuration

For secure HBase you should also run a secure Zookeeper (see Zookeeper Secure Configuration above). If you do so you will need to execute the steps in this section. These steps must be done on the HBase master and all region servers.

Note: This is not fully tested due to Java DNS resolution issues with the test environment.

1. Create a file `/etc/gphd/hbase/conf/jaas.conf` and add the following to it:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  useTicketCache=false
  keyTab="/etc/security/keytab/hbase.service.keytab"
  principal="hbase/hostname@REALM";
};
```

Make sure to replace `hostname@REALM` with the hostname of the server and the correct REALM.

2. Add the following near at the end of `/etc/gphd/hbase/conf/hbase-env.sh`:

```
export HBASE_OPTS="$HBASE_OPTS
-Djava.security.auth.login.config=/etc/gphd/hbase/conf/jaas.conf"
export HBASE_MANAGES_ZK=false
```

3. Edit the site xml and add the following:

```
<property>
   <name>hbase.zookeeper.quorum</name>
   <value>comma-separated-list-of-zookeeper-hosts</value>
</property>

<property>
   <name>hbase.cluster.distributed</name>
   <value>true</value>
</property>
```

4. Edit `/etc/gphd/zookeeper/conf/zoo.cfg` and add the following:

```
kerberos.removeHostFromPrincipal=true
kerberos.removeRealmFromPrincipal=true
```

Hive Secure Configuration

The Hive MetaStore supports Kerberos authentication for Thrift clients. You can configure a standalone Hive MetaStoreServer instance to force clients to authenticate with Kerberos by setting the property hive.metastore.sasl.enabled property in the hive-default.xml configuration file to true, as shown in the example below.

Add the Kerberos principals and their locations to the `hive-default.xml` or `hive-site.xml` (if you are user). For example:

```
<property>
   <name>hive.metastore.sasl.enabled</name>
   <value>true</value>
   <description>If true, the metastore thrift interface will be
   secured with SASL. Clients must authenticate with
   Kerberos.</description>
</property>
<property>
   <name>hive.metastore.kerberos.keytab.file</name>
   <value>/etc/*****/hive.keytab</value>
   <description>The path to the Kerberos Keytab file containing the
   metastore thrift server's service principal.</description>
   </property>
<property>
   <name>hive.metastore.kerberos.principal</name>
   <value>hive-metastore/_HOST@EXAMPLE.COM</value>
   <description>The service principal for the metastore thrift
   server. The special string _HOST will be replaced automatically
   with the correct host name.</description>
</property>
```

MapReduce Version 1 Configuration (MRv1)

To configure MRv1 follow the same steps as for HDFS and YARN except for the following differences.

MRv1 Kerberos Set-up Differences

For MRV1 do not create the yarn principals. Instead create the mapred principal for all task tracker and job tracker hosts.

- MAPRED (JobTrackers, TaskTrackers): for each service host you need to do:
  `addprinc -randkey  mapred/hostname@REALM`

You will not need to create and distribute yarn keytab files, but you will instead have to create and distribute the mapred keytab files for the JobTracker and TaskTracker hosts.

Container and Script Modifications Differences

1. Instead of editing the `/usr/lib/gphd/hadoop-yarn/etc/hadoop/container-executor.cfg` file, edit the `/usr/lib/gphd/hadoop-mr1/etc/hadoop/task-controller.cfg` file to be:

   ```
   # NOTE: the "hadoop.log.dir" should be the same value as what
   the Hadoop daemons are using
   hadoop.log.dir=/var/log/gphd/hadoop-mr1
   mapreduce.tasktracker.group=mapred
   banned.users=mapred,hdfs,bin
   min.user.id=500
   ```

   Note: The `min.user.id` varies by Linux dist; for CentOS it is 500, RedHat is 1000.

2. Check the permissions on `/usr/lib/gphd/hadoop-mr1/bin/task-controller`. They should look like:

   ```
   ----Sr-s--- 1 root mapred   286 Jun 18 00:08 task-controller
   ```

   If they do not then set the owner, group, and permissions as:

```
chown root:mapred task-controller
chmod 050 task-controller
chmod u+s task-controller
chmod g+s task-controller
```

MRv1 Site XML Differences

For MRv1 you do not edit the `yarn-site.xml` file. Instead add the following to the `mapred-site.xml` file:

```
<!-- JobTracker configuration info -->
<!-- JobTracker principal must be known to all cluster hosts
-->
<property>
   <name>mapreduce.jobtracker.kerberos.principal</name>
   <value>mapred/jobtracker-hostname@REALM</value>
</property>

<!-- keytab only needs to be know to the JobTracker host -->
<property>
   <name>mapreduce.jobtracker.keytab.file</name>
   <value>/var/local/hadoop/mapred.service.keytab</value>
</property>

<!-- TaskTracker configuration info -->
<!-- TaskTracker principal must be known to all cluster
hosts -->
<property>
   <name>mapreduce.tasktracker.kerberos.principal</name>
   <value>mapred/this-tasktracker-hostname@REALM</value>
</property>

<property>
   <name>mapreduce.tasktracker.keytab.file</name>
   <value>/var/local/hadoop/mapred.service.keytab</value>
</property>

<!-- TaskController configuration info -->
<property>
   <name>mapred.task.tracker.task-controller</name>

   <value>org.apache.hadoop.mapred.LinuxTaskController</valu
   e>
</property>
```

```
<property>
   <name>mapreduce.tasktracker.group</name>
   <value>mapred</value>
?</property>
```

Troubleshooting

- A good first step is to look for exceptions that may give you a hint in the logs (where `hostname` is the host on which the log file is located):
  - namenode: `/var/log/gphd/hadoop/hadoop-hdfs-namenode-hostname.log`
  - resourcemanager: `/var/log/gphd/hadoop-yarn/yarn-yarn-resourcemanager-hostname.log`
  - historyserver: `/var/log/gphd/hadoop-mapreduce/mapred-mapred-historyserver-hostname.log`
  - datanode: `/var/log/gphd/hadoop-hdfs/hdfs/hadoop-hdfs-datanode-hostname.log`
  - nodemanager: `/var/log/gphd//hadoop-yarn/yarn-yarn-nodemanager-hostname.log`

  You can enable debug level logging for the Java Kerberos classes by editing `/etc/default/hadoop` and setting the following:

  ```
  HADOOP_OPTS="$HADOOP_OPTS -Dsun.security.krb5.debug=true"
  ```
- Datanode will not start:
  - If you are getting a message about datanode requiring privileged resources to start check your port are < 1024 in `yarn-site.xml`.
  - Make sure you only changed the ports indicated in the instructions to be < 1024.
  - Make sure `core-site.xml` is configured to use kerberos.
  - Check keytab and principal entries in site xml, keytab directory owner/group is correct.
  - To inspect keytab files use: `klist -e -k -t pathtokeytab`.
  - Check that you modified `hadoop-env.sh` and `/etc/init.d/hadoop-hdfs-datanode` properly.
  - If the above checks are OK, run `service hadoop-hdfs-datanode start` and look at the output.
  - If there are no printed errors or you see an error stating that no VM can be found, it is a JSVC problem, see "Building and Installing JSVC" on page 95.
- Unable to find principal:
  - Check keytab and principal entries in site xml, keytab dir permissions.

- Unable to get password for username:
  - Check keytab and principal entries in site xml, keytab directory permissions. If these all look OK then run:

    ```
    kinit -k -t
    /etc/security/keytab/servicename.service.keytab
    ```

    You should get no errors (just a prompt back). If there is an error check that the principal and keytab are correct.
  - Check to make sure you used `-norandkey` when creating keytab files.
- Node manager will not start:
  - Login failure due to policy error exceptions in logs (typically seen as a remote exception to node manager for resource manager): check `/usr/lib/gphd/hadoop/etc/hadoop/hadoop-policy.xml` and replace any occurrences of `${HADOOP_HDFS_USER}` with hdfs and `${HADOOP_YARN_USER}` with yarn.

A. Creating a YUM EPEL Repository

Pivotal Command Center and Pivotal HD Enterprise expect some prerequisite packages to be pre-installed on each host, depending on the software that gets deployed on a particular host. In order to have a smoother installation it is recommended that each host would have yum access to an EPEL yum repository. If you have access to the Internet, then you can configure your hosts to have access to the external EPEL repositories. However, if your hosts do not have Internet access (or you are deploying onto a large cluster), then having a local yum EPEL repo would be highly recommended. This will also give you some control on the package versions you want deployed on your cluster.

Following are the steps to create a local yum repo:

1. Mount the RHEL/CentOS DVD on a machine that will act as the local yum repo
2. Install a webserver on that machine (e.g. httpd), making sure that HTTP traffic can reach this machine
3. Install the following packages on the machine:
   ```
   yum-utils
   createrep
   ```
4. Go to the directory where the DVD is mounted and run the following command:
   ```
   # createrepo .
   ```
5. Create a repo file on each host with a descriptive filename in the `/etc/yum.repos.d/` directory of each host (for example, *`CentOS-6.1.repo`*) with the following contents:
   ```
   [CentOS-6.1]
   name=CentOS 6.1 local repo for OS RPMS
   baseurl=http://172.254.51.221/centos/$releasever/os/
   $basearch/
   enabled=1
   gpgcheck=1
   gpgkey=http://172.254.51.221/centos/$releasever/os/$basearch
   /RPM-GPG-KEY-CentOS-6
   ```
6. Validate that you can access the local yum repos by running the following command:
   ```
   Yum list
   ```