

# Pivotal HD Enterprise

Version 1.1

## Stack and Tool Reference Guide

Rev: A01

# Table of Contents

1	Pivotal HD 1.1 Stack - RPM Package	11
1.1	Overview	11
1.2	Accessing PHD 1.1	11
1.2.1	Prerequisites	12
1.3	Hadoop HDFS	12
1.3.1	Hadoop HDFS RPM Packages	13
1.3.2	Prerequisites: Core Package Setup	16
1.3.3	HDFS Namenode Setup	16
1.3.4	HDFS Datanode Setup	16
1.3.5	HDFS Secondary Namenode Setup	16
1.3.6	Hadoop HDFS Configuration	17
1.3.7	Usage	17
1.4	Hadoop YARN	19
1.4.1	Hadoop YARN RPM Packages	19
1.4.2	Prerequisites: Core Package Setup	22
1.4.3	YARN ResourceManager Setup	22
1.4.4	YARN NodeManager Setup	22
1.4.5	Mapreduce HistoryServer Setup	22
1.4.6	YARN ProxyServer Setup	23
1.4.7	Configuration	23
1.4.8	YARN Usage	23
1.5	Zookeeper	26
1.5.1	Zookeeper RPM Packages	26
1.5.2	Zookeeper Server Setup	27
1.5.3	Zookeeper Client Setup	27
1.5.4	Zookeeper Configuration	27
1.5.5	Usage	27
1.6	HBase	28
1.6.1	Prerequisites	28
1.6.2	HBase RPM Packages	28
1.6.3	HBase Master Setup	30
1.6.4	HBase RegionServer Setup	30
1.6.5	HBase Client Setup	30
1.6.6	HBase Thrift Server Setup	30
1.6.7	REST Server Setup	30
1.6.8	HBase Configuration	31
1.6.9	HBase Post-Installation Configuration	31
1.6.10	Usage	31
1.7	Hive	34
1.7.1	Hive Components	34
1.7.2	Prerequisites	34
1.7.3	Hive RPM Packages	34

1.7.4	Installing Hive	35
1.7.5	Hive Client Setup	38
1.7.6	Hive Thrift Server Setup	39
1.7.7	Hive Server2 Setup	39
1.7.8	Hive MetaStore Server Setup	39
1.7.9	Hive Configuration	39
1.7.10	Hive Post-installation Configuration	40
1.7.11	Hive Usage	40
1.8	Hcatalog	41
1.8.1	Prerequisites	41
1.8.2	Hcatalog RPM Packages	42
1.8.3	Hcatalog Client Setup	43
1.8.4	Hcatalog Server Setup	43
1.8.5	Webhcat Setup	43
1.8.6	Webhcat Server Setup	43
1.8.7	Hcatalog Configuration	43
1.8.8	Usage	44
1.9	Pig	45
1.9.1	Prerequisites	45
1.9.2	Pig RPM Packages	45
1.9.3	Pig Client Setup	45
1.9.4	Pig Configuration	46
1.9.5	Usage	46
1.10	Mahout	46
1.10.1	Prerequisites	46
1.10.2	Mahout RPM Packages	46
1.10.3	Mahout Client Setup	47
1.10.4	Mahout Configuration	47
1.10.5	Usage	47
1.11	Flume	47
1.11.1	Prerequisites	48
1.11.2	Flume RPM Packages	48
1.11.3	Flume Client Setup	48
1.11.4	Flume Agent Setup	49
1.11.5	Flume Configuration	49
1.11.6	Usage	49
1.12	Sqoop	50
1.12.1	Prerequisites	50
1.12.2	Sqoop RPM Packages	50
1.12.3	Sqoop Client Setup	51
1.12.4	Sqoop Metastore Setup	51
1.12.5	Sqoop Configuration	51
1.12.6	Usage	51
1.13	Oozie	52
1.13.1	Prerequisites	52
1.13.2	Oozie RPM Packages	52

1.13.3 Oozie client Setup	53
1.13.4 Oozie server Setup [Optional]	53
1.13.5 Oozie Configuration	53
1.13.6 Usage	53
2 Pivotal HD MR1 1.1 Stack - RPM Package	56
2.1 Overview	56
2.2 Accessing PHDMR1 1.1	56
2.2.1 Prerequisites	57
2.2.2 Installation Notes	57
2.3 Hadoop HDFS	58
2.3.1 Hadoop HDFS RPM Packages	58
2.3.2 Prerequisites: Core Package Setup	61
2.3.3 HDFS Namenode Setup	61
2.3.4 HDFS Datanode Setup	61
2.3.5 HDFS Secondary Namenode Setup	61
2.3.6 Hadoop HDFS Configuration	62
2.3.7 Usage	62
2.4 Hadoop MR1	64
2.4.1 Core Package Setup	65
2.4.2 Hadoop-mr1 JobTracker Setup	65
2.4.3 Hadoop-mr1 TaskTracker Setup	66
2.4.4 Hadoop MR1 Configuration	66
2.4.5 Usage	66
2.4.6 Using Hadoop-mr1	67
2.5 Zookeeper	68
2.5.1 Zookeeper RPM Packages	68
2.5.2 Zookeeper Server Setup	69
2.5.3 Zookeeper Client Setup	69
2.5.4 Zookeeper Configuration	70
2.5.5 Usage	70
2.6 HBase	70
2.6.1 Prerequisites	71
2.6.2 HBase RPM Packages	71
2.6.3 HBase Master Setup	72
2.6.4 HBase RegionServer Setup	72
2.6.5 HBase Client Setup	72
2.6.6 HBase Thrift Server Setup	73
2.6.7 REST Server Setup	73
2.6.8 HBase Configuration	73
2.6.9 HBase Post-Installation Configuration	73
2.6.10 Usage	74
2.7 Hive	76
2.7.1 Hive Components	76
2.7.2 Prerequisites	76
2.7.3 Hive RPM Packages	77
2.7.4 Installing Hive	78

2.7.5	Hive Client Setup	80
2.7.6	Hive Thrift Server Setup	81
2.7.7	Hive Server2 Setup	81
2.7.8	Hive MetaStore Server Setup	81
2.7.9	Hive Configuration	81
2.7.10	Hive Post-installation Configuration	81
2.7.11	Hive Usage	82
2.8	Hcatalog	83
2.8.1	Prerequisites	83
2.8.2	Hcatalog RPM Packages	84
2.8.3	Hcatalog Client Setup	85
2.8.4	Hcatalog Server Setup	85
2.8.5	Webhcat Setup	85
2.8.6	Webhcat Server Setup	85
2.8.7	Hcatalog Configuration	85
2.8.8	Usage	86
2.9	Pig	87
2.9.1	Prerequisites	87
2.9.2	Pig RPM Packages	87
2.9.3	Pig Client Setup	87
2.9.4	Pig Configuration	88
2.9.5	Usage	88
2.10	Mahout	88
2.10.1	Prerequisites	88
2.10.2	Mahout RPM Packages	88
2.10.3	Mahout Client Setup	89
2.10.4	Mahout Configuration	89
2.10.5	Usage	89
2.11	Flume	89
2.11.1	Prerequisites	90
2.11.2	Flume RPM Packages	90
2.11.3	Flume Client Setup	90
2.11.4	Flume Agent Setup	91
2.11.5	Flume Configuration	91
2.11.6	Usage	91
2.12	Sqoop	92
2.12.1	Prerequisites	92
2.12.2	Sqoop RPM Packages	92
2.12.3	Sqoop Client Setup	92
2.12.4	Sqoop Metastore Setup	93
2.12.5	Sqoop Configuration	93
2.12.6	Usage	93
3	Pivotal HD 1.1 Stack - Binary Package	95
3.1	Overview	95
3.2	Accessing PHD 1.1 Stack Binary Package	95
3.2.1	Version Table	95

3.3	Installation	96
3.3.1	Prerequisites	97
3.3.2	Hadoop	98
3.3.3	Zookeeper	102
3.3.4	HBase	103
3.3.5	Hive	105
3.3.6	HCatalog	105
3.3.7	Pig	107
3.3.8	Mahout	108
3.3.9	Sqoop	108
3.3.10	Flume	109
3.3.11	Oozie	110
4	Pivotal HD MR1 1.1 Stack - Binary Package	113
4.1	Overview	113
4.2	Accessing PHD MR1 1.1	113
4.3	Installation	114
4.3.1	Prerequisites	115
4.3.2	Hadoop	116
4.3.3	Zookeeper	119
4.3.4	HBase	120
4.3.5	Hive	122
4.3.6	HCatalog	123
4.3.7	Pig	124
4.3.8	Mahout	125
4.3.9	Sqoop	126
4.3.10	Flume	127
5	Pivotal HD 1.1 Stack - Other Components	129
5.1	Overview	129
5.2	Spring Data	129
5.2.1	Installing Spring Data Hadoop	129
5.2.2	Installing Spring Data Hadoop through RPM	130
5.2.3	Spring Data Hadoop	130
5.3	HDFS Rack Awareness	130
5.3.1	Usage	133
5.4	Vaidya	133
5.4.1	Overview	133
5.4.2	Installing Vaidya Files	134
5.4.3	Enabling and Disabling Vaidya	134
5.4.4	Using Vaidya to Analyze Jobs	135
5.4.5	Vaidya Configuration Rules	135
5.5	DataLoader	138
6	USS Documentation - Main	139
6.1	Overview	139
6.1.1	Versions	139
6.1.2	Supported Features	139
6.1.3	Architecture	140

6.1.4	Prerequisites	141
6.2	Getting Started	142
6.2.1	USS Namenode (Server)	142
6.2.2	USS Catalog	142
6.2.3	USS Agent	143
6.2.4	USS Client	143
6.2.5	USS Gateway	143
6.2.6	Package Contents	144
6.3	Installing USS	146
6.4	Configuring USS	146
6.4.1	USS Namenode Configuration	147
6.4.2	USS Client Configuration	148
6.4.3	USS Catalog Configuration	149
6.5	Using the USS Command Line Interface	150
6.5.1	Features	150
6.5.2	Usage	150
6.5.3	USS Mountpoint Operations	153
6.5.4	Initialize USS Catalog	154
6.5.5	USS Classpath	154
6.5.6	USS Version	154
6.6	Protocol-Incompatible Filesystem Support	154
6.6.1	Supported Protocol-Incompatible Filesystem	155
6.6.2	Passphraseless Access Setup	156
6.6.3	Register a Protocol-Incompatible Filesystem	156
6.6.4	Adding a Mount Point	158
6.6.5	Testing Access	158
6.7	Secure HDFS support	158
6.7.1	Secure HDFS and MountPoint Registration	158
6.8	Using USS	159
6.8.1	Setting up the environment	159
6.8.2	Testing USS	159
6.9	USS Namenode Service	163
6.9.1	Starting the Service	163
6.9.2	Stopping the Service	163
6.9.3	Restarting the Service	163
6.9.4	Checking the Service Status	163
6.10	Uninstallation	163
6.11	Troubleshooting/FAQs	164
6.12	Terminology	168
7	HVE (Hadoop Virtualization Extensions)	170
7.1	Overview	170
7.2	Topology Awareness	170
7.2.1	Topology Awareness Configuration and Verification	170
7.3	Elasticity	173
7.3.1	Overview	173
7.3.2	Function List	173

7.3.3	Configuration	173
7.3.4	Command Line Interface	174
8	Security	180
8.1	Security	180
8.2	Configuring Kerberos for HDFS and YARN (MapReduce)	180
8.2.1	Kerberos Set-up	180
8.2.2	Install Kerberos Workstation and Libraries on Cluster Hosts	181
8.2.3	Distribute the Kerberos Client Configuration File to all Cluster Hosts	181
8.2.4	Create the Principals	181
8.2.5	Create the Keytab Files	182
8.2.6	Distribute the Keytab Files	184
8.2.7	Java Support Items Installation	185
8.2.8	Container and Script Modifications	185
8.2.9	Site XML Changes	187
8.2.10	Complete the HDFS/YARN Secure Configuration	192
8.2.11	Turning Secure Mode Off	193
8.2.12	Building and Installing JSVC	193
8.2.13	Installing the MIT Kerberos 5 KDC	195
8.3	Zookeeper Secure Configuration	197
8.3.1	Zookeeper Servers	197
8.3.2	Zookeeper Clients	199
8.4	HBase Secure Configuration	200
8.4.1	HBase Master and RegionServers	200
8.4.2	HBase Clients	202
8.4.3	HBase with Secure Zookeeper Configuration	203
8.5	Hive Secure Configuration	204
8.6	USS Secure Configuration	205
8.6.1	Securing USS	205
8.7	MapReduce Version 1 Configuration (MRv1)	205
8.7.1	MRv1 Kerberos Set-Up Differences	205
8.7.2	Container and Script Modifications Differences	206
8.7.3	MRv1 Site XML Differences	206
8.8	Auditing	207
8.9	Secure Web Access	208
8.9.1	Overview	208
8.9.2	Prerequisites	209
8.9.3	Configuring Secure WebHDFS	210
8.9.4	Using WebHDFS in Secure Mode	210
8.10	Troubleshooting	212
9	Manually Upgrading PHD from 1.0.1 to 1.1 - RPM	214
9.1	Overview	214
9.2	Components to be Upgraded (ICM support)	214
9.2.1	Upgrade Bigtop utilities	214
9.2.2	Upgrade Zookeeper	214
9.2.3	Upgrade Hadoop	215
9.2.4	Upgrade HBase	215



9.2.5	Upgrade Hive	215
9.2.6	Upgrade Pig	215
9.2.7	Upgrade Mahout	215
9.3	Components to be Upgraded (ICM don't support)	215
9.4	Upgrade Instructions	216
9.4.1	Download and extract the new PHD 1.1 tarball	216
9.4.2	Backup old configurations	216
9.4.3	Upgrade Flume	216
9.4.4	Upgrade Sqoop	217
10	Manually Upgrading PHDMR1 from 1.0.1 to 1.1 - RPM	219
10.1	Overview	219
10.2	Components to be upgraded	219
10.3	Prerequisites	219
10.4	Upgrade Instructions	220
10.4.1	Download and extract the new PHDMR1 1.1 tarball	220
10.4.2	Backup old configurations	220
10.4.3	Upgrade Zookeeper	221
10.4.4	Upgrade Bigtop Utilities	222
10.4.5	Upgrade HDFS	222
10.4.6	Upgrade Hadoop MR1	224
10.4.7	Upgrade HBase	226
10.4.8	Upgrade Hive	228
10.4.9	Upgrade Pig	229
10.4.10	Upgrade Flume	229
10.4.11	Upgrade Sqoop	230
10.4.12	Upgrade Mahout	231
11	Manually Upgrading PHD/PHDMR1 from 1.0.1 to 1.1 - Binary	232
11.1	Overview	232
11.2	Components to be Upgraded	232
11.3	Prerequisites	232
11.4	Upgrade Zookeeper	232
11.5	Upgrade Hadoop	233
11.6	Upgrade HBase	233
11.7	Upgrade Hive	233
11.8	Upgrade Pig	234
11.9	Upgrade Flume	234
11.10	Upgrade Sqoop	234
11.11	Upgrade Mahout	234

Copyright © 2013 GoPivotal, Inc. All rights reserved.

GoPivotal, Inc. believes the information in this publication is accurate as of its publication date. The information is subject to change without notice. THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." GOPIVOTAL, INC. ("Pivotal") MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any Pivotal software described in this publication requires an applicable software license.

All trademarks used herein are the property of Pivotal or their respective owners.

### **Use of Open Source**

This product may be distributed with open source code, licensed to you in accordance with the applicable open source license. If you would like a copy of any such source code, Pivotal will provide a copy of the source code that is required to be made available in accordance with the applicable open source license. Pivotal may charge reasonable shipping and handling charges for such distribution.

# 1 Pivotal HD 1.1 Stack - RPM Package

## 1.1 Overview

Pivotal HD 1.1 is a full Apache Hadoop distribution with Pivotal add-ons and a native integration with the Pivotal Greenplum database.

The RPM distribution of PHD 1.1 contains the following:

- **Hadoop 2.0.5-alpha**
- **Pig 0.10.1**
- **Zookeeper 3.4.5**
- **HBase 0.94.8**
- **Hive 0.11.0**
- **Hcatalog 0.11.0**
- **Mahout 0.7**
- **Flume 1.3.1**
- **Sqoop 1.4.2**
- **Oozie 3.3.2**

## 1.2 Accessing PHD 1.1

You can download the package from EMC Download Center, e

```
$ tar zxvf PHD-1.1.0.0-nn.tar.gz
$ ls -l PHD-1.1.0.0-nn
total 40
drwxr-xr-x 3 500 500 4096 Jun 25 06:06 flume
drwxr-xr-x 3 500 500 4096 Jun 25 06:07 hadoop
drwxr-xr-x 3 500 500 4096 Jun 25 06:06 hbase
drwxr-xr-x 3 500 500 4096 Jun 25 06:06 hive
drwxr-xr-x 3 500 500 4096 Jun 25 06:06 hcatalog
drwxr-xr-x 3 500 500 4096 Jun 25 06:07 mahout
drwxr-xr-x 3 500 500 4096 Jun 25 06:07 oozie
drwxr-xr-x 3 500 500 4096 Jun 25 06:06 pig
-rw-rw-r-- 1 500 500 376 Jun 25 06:06 README
drwxr-xr-x 3 500 500 4096 Jun 25 06:07 sqoop
drwxr-xr-x 3 500 500 4096 Jun 25 06:06 utility
drwxr-xr-x 3 500 500 4096 Jun 25 06:07 zookeeper
```

We define the replaced string which will be used in the following sections for each component.

Component	PHD Version	Replaced String
Hadoop	2.0.5_alpha_gphd_2_1_0_0	<PHD_HADOOP_VERSION>

Component	PHD Version	Replaced String
HBase	0.94.8_gphd_2_1_0_0	<PHD_HBASE_VERSION>
Hive	0.11.0_gphd_2_1_0_0	<PHD_HIVE_VERSION>
Pig	0.10.1_gphd_2_1_0_0	<PHD_PIG_VERSION>
Mahout	0.7_gphd_2_1_0_0	<PHD_MAHOUT_VERSION>
HCatalog	0.10.1_gphd_2_1_0_0	<PHD_HCATALOG_VERSION>
Sqoop	1.4.2_gphd_2_1_0_0	<PHD_SQOOP_VERSION>
Flume	1.3.1_gphd_2_1_0_0	<PHD_FLUME_VERSION>
Zookeeper	3.4.5_gphd_2_1_0_0	<PHD_ZOOKEEPER_VERSION>
Oozie	3.3.2_gphd_2_1_0_0	<PHD_OOZIE_VERSION>
bigtop-jsvc	1.0.15_gphd_2_1_0_0	<PHD_BIGTOP_J SVC_VERSION>
bigtop-utils	0.4.0_gphd_2_1_0_0	<PHD_BIGTOP_UTILS_VERSION>

## Installation

This section describes how to manually install, configure, and use Pivotal HD 1.1.

### 1.2.1 Prerequisites

- Oracle Java Development Kit (JDK) 1.6 or 1.7. 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 section we install packages by running:

```
rpm ivh <package_name>.<version>.rpm
```

Within this documentation, `nn` within the rpm file names represents the rpm build number. It is different for different components.

## 1.3 Hadoop HDFS

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

- [HDFS Namenode Setup](#)
- [HDFS Datanode Setup](#)
- [HDFS Secondary Namenode Setup](#)

## 1.3.1 Hadoop HDFS 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 HDFS daemons as Linux services, which rely on the core package.

hadoop-<PHD_HADOOP_VERSION>-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-<PHD_HADOOP_VERSION>-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-<PHD_HADOOP_VERSION>-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-<PHD_HADOOP_VERSION>-nn.x86_64.rpm	
Type	Daemon

<b>Requires</b>	hadoop-hdfs
<b>Description</b>	Daemon scripts package for Hadoop Datanode, which provides a convenient method to manage datanode start/stop as a Linux service.
<b>Install on Nodes</b>	Install on all HDFS Datanodes.

**hadoop-hdfs-secondarynamenode-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

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

**hadoop-hdfs-journalnode-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

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

**hadoop-hdfs-zkfc-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

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

**hadoop-hdfs-fuse-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

<b>Type</b>	Core
<b>Requires</b>	hadoop-libhdfs, hadoop-client
<b>Description</b>	Binaries that can be used to mount hdfs as a local directory.

<b>Install on Nodes</b>	Install on the servers that want to mount the HDFS.
-------------------------	---

<b>hadoop-libhdfs-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Core
<b>Requires</b>	hadoop-hdfs
<b>Description</b>	Native implementation of the HDFS.
<b>Install on Nodes</b>	Install on servers that you want to run native hdfs.

<b>hadoop-httpfs-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Core
<b>Requires</b>	bigtop-tomcat, hadoop, hadoop-hdfs
<b>Description</b>	HttpFS is a server that provides a REST HTTP gateway supporting all HDFS File System operations (read and write).
<b>Install on Nodes</b>	Install on servers that will be serving REST HDFS service

<b>hadoop-doc-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Doc
<b>Requires</b>	N/A
<b>Description</b>	Document package provides the Hadoop document.
<b>Install on Nodes</b>	Install on whichever host that user want to read hadoop documentation.

<b>hadoop-conf-pseudo-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Configuration
<b>Requires</b>	hadoop-hdfs-datanode, hadoop-hdfs-secondarynamenode, hadoop-yarn-resourcemanager, hadoop-hdfs-namenode, hadoop-yarn-nodemanager, hadoop-mapreduce-historyserver
<b>Description</b>	A set of configuration files for running Hadoop in pseudo-distributed mode on one single server.
<b>Install on Nodes</b>	Install on the pseudo--distributed host.

<b>hadoop-client-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Library

<b>Requires</b>	N/A
<b>Description</b>	A set of symbolic link which gathers the libraries for programming Hadoop and submit Hadoop jobs.
<b>Install on Nodes</b>	Clients nodes that will be used to submit hadoop jobs.

## 1.3.2 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-<PHD_BIGTOP_UTILS_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/zookeeper/rpm/zookeeper-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
Where working_dir is the directory where you want the rpms expanded.
```

## 1.3.3 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-<PHD_BIGTOP_JSVC_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-namenode-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

## 1.3.4 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-<PHD_BIGTOP_JSVC_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-datanode-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

## 1.3.5 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-<PHD_BIGTOP_JSVC_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-secondarynamenode-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```



## 1.3.6 Hadoop HDFS Configuration

The configuration files for Hadoop are located here: `/etc/gphd/hadoop/conf/`

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

You can make modifications to these configuration templates or create your own configuration set. If you want to use a different configuration folder, edit 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 make sure all the dependent packages of `hadoop-conf-pseudo` have been installed on your host and then install the `hadoop-conf-pseudo` package:

```
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-conf-pseudo-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

Refer to Apache Hadoop 2.0.5-alpha documentation for how to configure Hadoop in distributed mode. This documentation describes how to use Hadoop in a pseudo-distributed mode.

## 1.3.7 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
```

## 1.4 Hadoop YARN

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

- [YARN ResourceManager Setup](#)
- [YARN NodeManager Setup](#)
- [Mapreduce HistoryServer Setup](#)
- [YARN ProxyServer Setup](#)

### 1.4.1 Hadoop YARN 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 YARN daemons as Linux services, which rely on the core package.

**hadoop-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

<b>Type</b>	Core
<b>Requires</b>	bigtop-utils, zookeeper-core
<b>Description</b>	Hadoop core packages provides the common core packages for running Hadoop.
<b>Install on Nodes</b>	Every node in the Hadoop cluster and the client workstation that will access the Hadoop service.

**hadoop-yarn-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

<b>Type</b>	Core
<b>Requires</b>	hadoop
<b>Description</b>	Hadoop YARN core packages provides common files for running YARN.
<b>Install on Nodes</b>	Install on all YARN nodes.

**hadoop-yarn-resourcemanager-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

<b>Type</b>	Daemon
<b>Requires</b>	hadoop-yarn
<b>Description</b>	Daemon scripts package for Hadoop YARN ResourceManager, which provides a convenient method to manage ResourceManager start/stop as a Linux service.
<b>Install on Nodes</b>	Install on the Resource Manager node.

**hadoop-yarn-nodemanager-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

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

**hadoop-yarn-proxyserver-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

<b>Type</b>	Daemon
<b>Requires</b>	hadoop-yarn

<b>Description</b>	Daemon scripts package for Hadoop YARN ProxyServer, which provides a convenient method to manage ProxyServer start/stop as a Linux service.
<b>Install on Nodes</b>	Install on the node that will act as a proxy server from the user to applicationmaster

hadoop-mapreduce-<PHD_HADOOP_VERSION>-nn.x86_64.rpm	
<b>Type</b>	Core
<b>Requires</b>	hadoop-yarn
<b>Description</b>	Hadoop Mapreduce core libraries.
<b>Install on Nodes</b>	Install on all ResourceManager and NodeManager nodes.

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

hadoop-doc-<PHD_HADOOP_VERSION>-nn.x86_64.rpm	
<b>Type</b>	Doc
<b>Requires</b>	N/A
<b>Description</b>	Document package provides the Hadoop documentation.
<b>Install on Nodes</b>	Install on whichever host that user want to read hadoop doc.

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

<b>hadoop-client-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Library
<b>Requires</b>	hadoop, hadoop-hdfs, hadoop-yarn, hadoop-mapreduce
<b>Description</b>	A set of symbolic link which gathers the libraries for programming Hadoop and submit Hadoop jobs.
<b>Install on Nodes</b>	Clients nodes that will be used to submit hadoop jobs.

## 1.4.2 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-<PHD_BIGTOP_UTILS_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/zookeeper/rpm/zookeeper-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

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

## 1.4.3 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-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-yarn-resourcemanager-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

## 1.4.4 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-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-yarn-nodemanager-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

## 1.4.5 Mapreduce HistoryServer Setup

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

```
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-yarn-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-mapreduce-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-mapreduce-historyserver-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

## 1.4.6 YARN ProxyServer Setup

Install the YARN Proxy Server package and its dependency packages on the workstation that will serve as the YARN Proxy Server.

```
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-yarn-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh
working_dir/hadoop/rpm/hadoop-yarn-proxyserver-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

## 1.4.7 Configuration

The configuration files for Hadoop are located here: `/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-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

Refer to Apache Hadoop 2.0.5-alpha documentation for how to configure Hadoop in distributed mode. This document describes how to use Hadoop in a pseudo-distributed mode.

## 1.4.8 YARN Usage

### 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 on HDFS, as follows:

**Create working directories on HDFS:**

```
$ sudo -u hdfs hdfs dfs -mkdir /tmp
$ sudo -u hdfs hdfs dfs -chmod 777 /tmp
$ sudo -u hdfs hdfs dfs -mkdir -p /var/log/gphd/hadoop-yarn
$ sudo -u hdfs hdfs dfs -chown yarn:hadoop /var/log/gphd/hadoop-yarn
$ sudo -u hdfs hdfs dfs -mkdir -p /user/history
$ sudo -u hdfs hdfs dfs -chown mapred:hadoop /user/history
$ sudo -u hdfs hdfs dfs -chmod -R 777 /user/history
$ sudo -u hdfs hdfs dfs -mkdir -p /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.

**Note:** Make sure HDFS daemons are running.

Here is an example MapReduce job:

```
$ hadoop jar /usr/lib/gphd/hadoop-mapreduce/hadoop-mapreduce-examples-*.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-*.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
```

## 1.5 Zookeeper

The base version of ZooKeeper is Apache ZooKeeper 3.4.5.

ZooKeeper is a high-performance coordination service for distributed applications.

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

### 1.5.1 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-<PHD_ZOOKEEPER_VERSION>-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-<PHD_ZOOKEEPER_VERSION>-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-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm	
Type	Documentation
Requires	N/A

Description	Zookeeper documentation.
Install on Nodes	N/A

## 1.5.2 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-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/zookeeper/rpm/zookeeper-server-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
```

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

## 1.5.3 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-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
```

## 1.5.4 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.

## 1.5.5 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:

## 1.6 HBase

The base version of HBase changed to Apache HBase 0.94.8.

HBase is a scalable, distributed database that supports structured data storage for large tables.

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

### 1.6.1 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.

### 1.6.2 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

<b>Type</b>	Core
<b>Requires</b>	Hadoop HDFS Packages and ZooKeeper Core Package
<b>Description</b>	HBase core package provides all executables, libraries, configuration files and documentations.

**hbase-master-<PHD\_HBASE\_VERSION>-nn.noarch.rpm**

<b>Type</b>	Daemon
<b>Requires</b>	HBase Core Package
<b>Description</b>	Daemon scripts package for HMaster, which provides a convenient method to manage HBase HMaster server start/stop as a Linux service.

**hbase-regionserver-<PHD\_HBASE\_VERSION>-nn.noarch.rpm**

<b>Type</b>	Daemon
<b>Requires</b>	HBase Core Package
<b>Description</b>	Daemon scripts package for HRegionServer, which provides a convenient method to manage HBase HRegionServer start/stop as a Linux service.

**hbase-thrift-<PHD\_HBASE\_VERSION>-nn.noarch.rpm**

<b>Type</b>	Daemon (thrift service)
<b>Requires</b>	HBase Core Package
<b>Description</b>	Daemon scripts package to provide HBase service through thrift.

**hbase-rest-<PHD\_HBASE\_VERSION>-nn.noarch.rpm**

<b>Type</b>	Daemon (Restful service)
<b>Requires</b>	HBase Core Package
<b>Description</b>	Daemon scripts package to provide HBase service through REST.

**hbase-doc-<PHD\_HBASE\_VERSION>-nn.noarch.rpm**

<b>Type</b>	Documentation
<b>Requires</b>	HBase Core Package
<b>Description</b>	HBase documentation.

## 1.6.3 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hbase/rpm/hbase-master-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

## 1.6.4 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hbase/rpm/hbase-regionserver-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

## 1.6.5 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

## 1.6.6 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hbase/rpm/hbase-thrift-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

## 1.6.7 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hbase/rpm/hbase-rest-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

## 1.6.8 HBase Configuration

The configuration files for HBase are located here: `/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.

## 1.6.9 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 `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
```

## 1.6.10 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
```

## 1.7 Hive

The base version of Hive is Apache Hive 0.11.0.

Hive is a data warehouse infrastructure that provides data summarization and ad hoc querying.

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

### 1.7.1 Hive Components

A Hive installation consists of the following components:

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

### 1.7.2 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:

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

```
hive/gphd/warehouse
```

```
hive.metastore.local = false
```

### 1.7.3 Hive RPM Packages

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

hive-<PHD_HIVE_VERSION>-nn.noarch.rpm	
Type	Core
Requires	Hadoop, HBase Core Packages
Description	Hive core package provides the executables, libraries, configuration files and documentations.

<b>Install on Nodes</b>	Hive Client workstation
-------------------------	-------------------------

<b>hive-server-&lt;PHD_HIVE_VERSION&gt;-nn.noarch.rpm</b>	
<b>Type</b>	Daemon (thrift server)
<b>Requires</b>	Hive Core Package
<b>Description</b>	Daemon scripts package to provide Hive service through thrift
<b>Install on Nodes</b>	Hive Thrift server node

<b>hive-metastore-&lt;PHD_HIVE_VERSION&gt;-nn.noarch.rpm</b>	
<b>Type</b>	Deamon (Metastore server)
<b>Requires</b>	Hive Core Package
<b>Description</b>	Daemon scripts package to provide Hive metadata information through metastore server.
<b>Install on Nodes</b>	Hive Metastore server node

<b>hive-server2-&lt;PHD_HIVE_VERSION&gt;-nn.noarch.rpm</b>	
<b>Type</b>	Daemon (hive server2)
<b>Requires</b>	Hive Core Package
<b>Description</b>	Daemon scripts package to provide Hive Server2.
<b>Install on Nodes</b>	Hive Thrift server node

## 1.7.4 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

```
$ sudo 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:

```
$ sudo yum install postgresql-jdbc  
$ sudo rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm  
$ sudo rpm -ivh working_dir/hive/rpm/hive-metastore-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

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.war</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-*.jar:\
$HIVELIB_HOME/libthrift-0.7.0.jar:\
$HIVELIB_HOME/hive-metastore-*.jar:\
$HIVELIB_HOME/libfb303-0.7.0.jar:\
$HIVELIB_HOME/hive-common-*.jar:\
$HIVELIB_HOME/hive-exec-*.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_2_0/scripts/metastore/upgrade/postgres/hive-schema-0.9.0.p
```

7. Start the hive-metastore:

```
$ service hive-metastore start
```

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

## 1.7.5 Hive Client Setup

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

```
$ sudo rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

## 1.7.6 Hive Thrift Server Setup

### [OPTIONAL]

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

```
$ sudo rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hive/rpm/hive-server-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

## 1.7.7 Hive Server2 Setup

### [OPTIONAL]

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

```
$ sudo rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hive/rpm/hive-server2-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

## 1.7.8 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 rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hive/rpm/hive-metastore-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

## 1.7.9 Hive Configuration

The configuration files for Hive are located here: `/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.

## 1.7.10 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
```

## 1.7.11 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 Beeline Client

HiveServer2 supports a new command shell Beeline that works with HiveServer2:

```
$ beeline
```

### 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 Server2

[Optional]

You can start/stop Hive server2 daemon as follows:

Run:

```
$ sudo service hive-server2 start
$ sudo service hive-server2 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 [Security](#) section to configure Hive for a security-enabled HD cluster.

## 1.8 Hcatalog

The base version of Hcatalog is Apache Hcatalog 0.11.0.

HCatalog is a metadata and table management system.

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

### 1.8.1 Prerequisites

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

## 1.8.2 Hcatalog RPM Packages

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

hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm	
<b>Type</b>	Core
<b>Requires</b>	Hadoop, HBase and Hive Core Packages.
<b>Description</b>	Hcatalog core package provides the executables, libraries, configuration files and documentations.
<b>Install on Nodes</b>	Hcatalog Client workstation.

hcatalog-server-<PHD_HCATALOG_VERSION>-nn.noarch.rpm	
<b>Type</b>	Daemon (hcatalog server).
<b>Requires</b>	Hcatalog Core Package.
<b>Description</b>	Daemon scripts package to provide Hive service through thrift.
<b>Install on Nodes</b>	Hcatalog server node.

webhcat-<PHD_HCATALOG_VERSION>-nn.noarch.rpm	
<b>Type</b>	Libraries.
<b>Requires</b>	Hcatalog Core Package.
<b>Description</b>	Daemon scripts package to provide Hive metadata information through metastore server.
<b>Install on Nodes</b>	Webhcat server node.

webhcat-server-<PHD_HCATALOG_VERSION>-nn.noarch.rpm	
<b>Type</b>	Daemon(webhcata server).
<b>Requires</b>	Hcatalog and Webhcat Core Package.
<b>Description</b>	Daemon scripts package to provide Webhcat Server.

<b>Install on Nodes</b>	Webhcat server node.
-------------------------	----------------------

## 1.8.3 Hcatalog Client Setup

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

```
$ sudo rpm -ivh working_dir/hive/rpm/hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
```

## 1.8.4 Hcatalog Server Setup

[OPTIONAL]

Install the Hcatalog core package and Hcatalog thrift daemon package to provide Hcatalog service.

```
$ sudo rpm -ivh working_dir/hcatalog/rpm/hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hcatalog/rpm/hcatalog-server-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
```

## 1.8.5 Webhcat Setup

[OPTIONAL]

Install the Hcatalog core package and Webhcat package to provide Webhcat libraries.

```
$ sudo rpm -ivh working_dir/hcatalog/rpm/hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hcatalog/rpm/webhcat-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
```

## 1.8.6 Webhcat Server Setup

[OPTIONAL]

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

```
$ sudo rpm -ivh working_dir/hcatalog/rpm/hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hcatalog/rpm/webhcat-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hcatalog/rpm/webhcat-server-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
```

## 1.8.7 Hcatalog Configuration

The configuration files for Hcatalog are located here: `/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.

## 1.8.8 Usage

### Start Hcatalog Client

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

```
$ hcat
```

You can check the `hive` command usage by running:

```
$ hcat -help
```

### Start/Stop Hcatalog Server

You can start/stop Hcatalog server daemon as follows:

Either:

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

or:

```
$ sudo /etc/init.d/hcatalog-server start
$ sudo /etc/init.d/hcatalog-server stop
```

### Start/Stop Webhcat Server

You can start/stop Webhcat server daemon as follows:

Either:

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

or:

```
$ sudo /etc/init.d/webhcat-server start
$ sudo /etc/init.d/webhcat-server stop
```

## 1.9 Pig

The base version of Pig is Apache Pig 0.10.1.

Pig is a high-level data-flow language and execution framework for parallel computation.

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

### 1.9.1 Prerequisites

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

### 1.9.2 Pig RPM Packages

Pig has only one core package.

pig-<PHD_PIG_VERSION>-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-<PHD_PIG_VERSION>-nn.noarch.rpm	
Type	Documentation
Requires	N/A
Description	Pig documentation.
Install on Nodes	N/A

### 1.9.3 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-<PHD_PIG_VERSION>-nn.noarch.rpm
```

## 1.9.4 Pig Configuration

The configuration files for Pig are located here: `/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.

## 1.9.5 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
```

## 1.10 Mahout

The base version of Mahout is Apache Mahout 0.7.

Mahout is a scalable machine learning and data mining library.

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

### 1.10.1 Prerequisites

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

### 1.10.2 Mahout RPM Packages

Mahout has only one core package.

mahout-<PHD_MAHOUT_VERSION>-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

### 1.10.3 Mahout 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-<PHD_MAHOUT_VERSION>-nn.noarch.rpm
```

### 1.10.4 Mahout 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.

### 1.10.5 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
```

## 1.11 Flume

The base version of Flume is Apache Flume 1.3.1.

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.

## 1.11.1 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)

## 1.11.2 Flume RPM Packages

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

flume-<PHD_FLUME_VERSION>-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-<PHD_FLUME_VERSION>-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.

## 1.11.3 Flume Client Setup

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

```
$ sudo rpm -ivh working_dir/flume/rpm/flume-<PHD_FLUME_VERSION>-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`

## 1.11.4 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-<PHD_FLUME_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/flume/rpm/flume-agent-<PHD_FLUME_VERSION>-nn.noarch.rpm
```

## 1.11.5 Flume Configuration

The configuration files for Flume are located here: `/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.

## 1.11.6 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

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
```

## 1.12 Sqoop

The base version of Sqoop is Apache Sqoop 1.4.2.

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.

### 1.12.1 Prerequisites

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

### 1.12.2 Sqoop RPM Packages

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

sqoop-<PHD_SQOOP_VERSION>-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-<PHD_SQOOP_VERSION>-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

## 1.12.3 Sqoop Client Setup

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

```
$ sudo rpm -ivh working_dir/sqoop/rpm/sqoop-<PHD_SQOOP_VERSION>-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.

## 1.12.4 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-<PHD_SQOOP_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/sqoop/rpm/sqoop-metastore-<PHD_SQOOP_VERSION>-nn.noarch.rpm
```

## 1.12.5 Sqoop Configuration

The configuration files for Flume are located here: /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.

## 1.12.6 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

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
```

## 1.13 Oozie

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

### 1.13.1 Prerequisites

Oozie is built on top of Hadoop, so Hadoop packages must be installed to get Oozie running. See Hadoop section for Hadoop installation instructions, Oozie can manipulate hive job and pig job in the workflow. So if you want to use hive job or pig job in your workflow, the Hive and Pig packages must be installed. See Hive section and Pig section for the installation instructions.

### 1.13.2 Oozie RPM Packages

Oozie consists of a oozie-client rpm package and a oozie package. Oozie package depends on oozie-client package.

**Table 10 Oozie RPM package**

Package Name	Type	Requires	Description	Ins on No
oozie-client-<PHD_OOZIE_VERSION>-nn.noarch.rpm	oozie client	bigtop-util	Oozie client package provides oozie library and client binray to connect to Oozie service.	Oo ser noc and Oo clie noc

oozie-<PHD_OOZIE_VERSION>-nn.noarch.rpm	Daemon(Oozie server)	bigtop-tomcat, hadoop-client, oozie-client; hive(optional) pig(optional)	Daemon package to provide Oozie service.	Oo ser noc
---	----------------------	--	--	------------

### 1.13.3 Oozie client Setup

Install oozie-client package on the client host which submits workflows to Oozie service.

```
$ sudo rpm -ivh working_dir/oozie/rpm/oozie-client-<PHD_OOZIE_VERSION>-nn.noarch.rpm
```

NOTE: User "oozie" and group "oozie" are to be created with correct configuration: uid oozie, gid oozie. It is a non-login user.

### 1.13.4 Oozie server Setup [Optional]

Install the oozie-client package and oozie package to provide Oozie service. oozie has the dependency with oozie-client package.

```
$ sudo rpm -ivh working_dir/oozie/rpm/oozie-client-<PHD_OOZIE_VERSION>-nn.noarch.rpm$ sudo rpm -ivh working_dir/oozie/rpm/oozie-<PHD_OOZIE_VERSION>-nn.noarch.rpm
```

### 1.13.5 Oozie Configuration

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

/etc/gphd/oozie/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/oozie/ to point to the folder you want to utilize at runtime.

### 1.13.6 Usage

#### Start Sqoop Client

To run Oozie scripts on a client machine, use the command oozie with the sub-command directly in shell. Each sub-command may have different arguments.

```
$ oozie [sub-command]
```

You can check the oozie command usage by:

```
$ oozie help
```

## Setup Oozie server

Before starting Oozie service, follow these steps to initialize Oozie server.

1. add the following conf to hadoop configuration core-site.xml. Restart HDFS and Yarn

```
<property> <name>hadoop.proxyuser.oozie.hosts</name> <value>*</value>
</property><property> <name>hadoop.proxyuser.oozie.groups</name>
<value>*</value></property>
```

2. mkdir for user oozie on HDFS

```
$ sudo -u hdfs hdfs dfs -mkdir -p /user/oozie$ sudo -u hdfs hdfs dfs
-chown oozie:oozie /user/oozie
```

3. create oozie database.

```
$ sudo service oozie init
```

4. setup oozie tomcat war file

```
$ sudo -u oozie oozie-setup prepare-war -hadoop 2.x /usr/lib/gphd/hadoop
-extjs /tmp/ext-2.2/
```

5. setup sharelib for oozie service. Replace namenode-host with your name node hostname, and replace namenode-port with your name node port.

```
$ sudo -u oozie oozie-setup sharelib create -fs hdfs:
//${namenode-host}:${namenode-port} -locallib
/usr/lib/gphd/oozie/oozie-sharelib.tar.gz
```

## Start/Stop Oozie Server [Optional]

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

Either:

```
$ sudo service oozie start$ sudo service oozie stop$ sudo service oozie status
```

or:

```
$ sudo /etc/init.d/oozie start$ sudo /etc/init.d/oozie stop$ sudo  
/etc/init.d/oozie status
```

## 2 Pivotal HD MR1 1.1 Stack - RPM Package

### 2.1 Overview

Pivotal HD 1.1 is a full Apache Hadoop distribution with Pivotal add-ons and a native integration with the Pivotal Greenplum database.

The RPM distribution of PHDMR1 1.1 contains the following:

- **HDFS 2.0.5-alpha**
- **Hadoop MR1 1.0.3**
- **Pig 0.10.1**
- **Zookeeper 3.4.5**
- **HBase 0.94.8**
- **Hive 0.11.0**
- **Hcatalog 0.11.0**
- **Mahout 0.7**
- **Flume 1.3.1**
- **Sqoop 1.4.2**

### 2.2 Accessing PHDMR1 1.1

You can download the package from EMC Download Center, expand the package in your working\_dir:

```
$ tar zxvf PHDMR1-1.1.0.0-nn.tar.gz
$ ls -l PHDMR1-1.1.0.0-nn
total 44
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 flume
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hadoop
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hadoop-mr1
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hbase
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hive
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hcatalog
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 mahout
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 pig
-rw-rw-r-- 1 hadoop hadoop 406 Jun 26 04:38 README
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 sqoop
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 utility
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 zookeeper
```

We define the replaced string which will be used in the following sections for each component.

Component	PHD Version	Replaced String
Hadoop	2.0.5_alpha_gphd_2_1_0_0	<PHD_HADOOP_VERSION>



Component	PHD Version	Replaced String
MR1	1.0.3_gphd_2_1_0_0	<PHD_MR1_VERSION>
HBase	0.94.8_gphd_2_1_0_0	<PHD_HBASE_VERSION>
Hive	0.11.0_gphd_2_1_0_0	<PHD_HIVE_VERSION>
Pig	0.10.1_gphd_2_1_0_0	<PHD_PIG_VERSION>
Mahout	0.7_gphd_2_1_0_0	<PHD_MAHOUT_VERSION>
HCatalog	0.10.1_gphd_2_1_0_0	<PHD_HCATALOG_VERSION>
Sqoop	1.4.2_gphd_2_1_0_0	<PHD_SQOOP_VERSION>
Flume	1.3.1_gphd_2_1_0_0	<PHD_FLUME_VERSION>
Zookeeper	3.4.5_gphd_2_1_0_0	<PHD_ZOOKEEPER_VERSION>
bigtop-jsvc	1.0.15_gphd_2_1_0_0	<PHD_BIGTOP_J SVC_VERSION>
bigtop-utils	0.4.0_gphd_2_1_0_0	<PHD_BIGTOP_UTILS_VERSION>

## Installation

This section describes how to manually install, configure, and use Pivotal HD 1.1.

### 2.2.1 Prerequisites

- Oracle Java Development Kit (JDK) 1.7. 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
```

### 2.2.2 Installation Notes

In this section we install packages by running:

```
rpm ivh <package_name>.<version>.rpm
```

Within this documentation, `nn` within the rpm file names represents the rpm build number. It is different for different components.

## 2.3 Hadoop HDFS

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

- [HDFS Namenode Setup](#)
- [HDFS Datanode Setup](#)
- [HDFS Secondary Namenode Setup](#)

### 2.3.1 Hadoop HDFS 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 HDFS daemons as Linux services, which rely on the core package.

hadoop-<PHD_HADOOP_VERSION>-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-<PHD_HADOOP_VERSION>-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-<PHD_HADOOP_VERSION>-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-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

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

**hadoop-hdfs-secondarynamenode-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

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

**hadoop-hdfs-journalnode-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

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

**hadoop-hdfs-zkfc-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

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

**hadoop-hdfs-fuse-<PHD\_HADOOP\_VERSION>-nn.x86\_64.rpm**

<b>Type</b>	Core
<b>Requires</b>	hadoop-libhdfs, hadoop-client
<b>Description</b>	Binaries that can be used to mount hdfs as a local directory.
<b>Install on Nodes</b>	Install on the servers that want to mount the HDFS.

<b>hadoop-libhdfs-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Core
<b>Requires</b>	hadoop-hdfs
<b>Description</b>	Native implementation of the HDFS.
<b>Install on Nodes</b>	Install on servers that you want to run native hdfs.

<b>hadoop-httpfs-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Core
<b>Requires</b>	bigtop-tomcat, hadoop, hadoop-hdfs
<b>Description</b>	HttpFS is a server that provides a REST HTTP gateway supporting all HDFS File System operations (read and write).
<b>Install on Nodes</b>	Install on servers that will be serving REST HDFS service

<b>hadoop-doc-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Doc
<b>Requires</b>	N/A
<b>Description</b>	Document package provides the Hadoop document.
<b>Install on Nodes</b>	Install on whichever host that user want to read hadoop documentation.

<b>hadoop-conf-pseudo-&lt;PHD_HADOOP_VERSION&gt;-nn.x86_64.rpm</b>	
<b>Type</b>	Configuration
<b>Requires</b>	hadoop-hdfs-datanode, hadoop-hdfs-secondarynamenode, hadoop-yarn-resourcemanager, hadoop-hdfs-namenode, hadoop-yarn-nodemanager, hadoop-mapreduce-historyserver
<b>Description</b>	A set of configuration files for running Hadoop in pseudo-distributed mode on one single server.
<b>Install on Nodes</b>	Install on the pseudo--distributed host.

hadoop-client-<PHD_HADOOP_VERSION>-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.

## 2.3.2 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-<PHD_BIGTOP_UTILS_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/zookeeper/rpm/zookeeper-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
Where working_dir is the directory where you want the rpms expanded.
```

## 2.3.3 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-<PHD_BIGTOP_JSVC_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-namenode-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

## 2.3.4 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-<PHD_BIGTOP_JSVC_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-datanode-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

## 2.3.5 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-<PHD_BIGTOP_JSVC_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-secondarynamenode-2.0.5_gphd_2_0_3_0-nn.x86_64.rpm
```

## 2.3.6 Hadoop HDFS Configuration

The configuration files for Hadoop are located here: `/etc/gphd/hadoop/conf/`

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

You can make modifications to these configuration templates or create your own configuration set. If you want to use a different configuration folder, edit 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 make sure all the dependent packages of `hadoop-conf-pseudo` have been installed on your host and then install the `hadoop-conf-pseudo` package:

```
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-conf-pseudo-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
```

Refer to Apache Hadoop 2.0.5-alpha documentation for how to configure Hadoop in distributed mode. This documentation describes how to use Hadoop in a pseudo-distributed mode.

## 2.3.7 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
```

## 2.4 Hadoop MR1

This section provides instructions for installing each of the following core Hadoop MR1 RPMs:

### Hadoop MR1 RPM Packages

hadoop-mr1-<PHD_MR1_VERSION>-nn.x86_64.rpm	
Type	Core
Requires	bigtop-utils, hdfs 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-mr1-jobtracker-<PHD_MR1_VERSION>-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-mr1
Description	Hadoop YARN core packages provides common files for running YARN.
Install on nodes	Install on the JobTracker node.

hadoop-mr1-tasktracker-<PHD_MR1_VERSION>-nn.x86_64.rpm	
Type	Daemon
Requires	hadoop-mr1
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 TaskTracker node.

hadoop-mr1-conf-pseudo-<PHD_MR1_VERSION>-nn.x86_64.rpm	
Type	Configuration
Requires	hadoop-mr1, hadoop-mr1-jobtracker, hadoop-mr1-tasktracker, hadoop-hdfs-datanode, hadoop-hdfs-secondarynamenode, hadoop-yarn-resourcemanager, hadoop-hdfs-namenode
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.

## 2.4.1 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-<PHD_BIGTOP_UTILS_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/zookeeper/rpm/zookeeper-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-<PHD_HADOOP_VERSION>-nn.x86_64.rpm
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-<PHD_HADOOP_VERSION> -nn.x86_64.rpm
```

## 2.4.2 Hadoop-mr1 JobTracker Setup

Install the Hadoop-mr1 JobTracker package on the workstation that will serve as the Hadoop-mr1 JobTracker:

```
$ sudo rpm -ivh working_dir/hadoop-mr1/rpm/hadoop-mr1-jobtracker-<PHD_MR1_VERSION>-nn.x86_64.rpm
```

## 2.4.3 Hadoop-mr1 TaskTracker Setup

Install the Hadoop-mr1 TaskTracker package on the workstation that will serve as the Hadoop-mr1 TaskTracker:

```
$ sudo rpm -ivh  
working_dir/hadoop-mr1/rpm/hadoop-mr1-tasktracker-<PHD_MR1_VERSION>-nn.x86_64.rpm
```

## 2.4.4 Hadoop MR1 Configuration

The configuration files for Hadoop MR1 are located here: `/etc/gphd/hadoop/conf/`

Out of the box by default it is a symbolic link to `/etc/gphd/hadoop-version/conf.empty` template 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 `/etc/gphd/hadoop/conf` symbolic link to point to the folder you want to utilize at runtime.

If you want to run Hadoop MR1 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-mr1-conf-pseudo` package:

```
$ sudo rpm -ivh  
working_dir/hadoop-mr1/rpm/hadoop-mr1-conf-pseudo-<PHD_MR1_VERSION>-nn.x86_64.rpm
```

## 2.4.5 Usage

### Starting Hadoop-MR1

Before you start Hadoop, you need to create some working directories on HDFS, as follows:

```
// 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
```

## Starting Hadoop-mr1 JobTracker

To start the hadoop-mr1 jobtracker daemon:

Either:

```
$ sudo service hadoop-mr1-jobtracker start
```

or:

```
$ sudo /etc/init.d/hadoop-mr1-jobtracker start
```

## Starting Hadoop-mr1 TaskTracker

To start the hadoop-mr1 tasktracker daemon:

Either:

```
$ sudo service hadoop-mr1-tasktracker start
```

or:

```
$ sudo /etc/init.d/hadoop-mr1-tasktracker start
```

## 2.4.6 Using Hadoop-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. In these examples we use the user `hadoop`.

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

Here is an example MapReduce job:

```
su - hadoop
$ hadoop-mr1 jar /usr/lib/gphd/hadoop-mr1-<PHD_MR1_VERSION>/hadoop-examples-*.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 Hadoop-mr1

### Stop Hadoop-mr1 JobTracker

To stop the hadoop-mr1 jobtracker daemon:

Either:

```
$ sudo service hadoop-mr1-jobtracker stop
```

or:

```
$ sudo /etc/init.d/hadoop-mr1-jobtracker stop
```

### Stop Hadoop-mr1 TaskTracker

To stop the hadoop-mr1 tasktracker daemon:

Either:

```
$ sudo service hadoop-mr1-tasktracker stop
```

or:

```
$ sudo /etc/init.d/hadoop-mr1-tasktracker stop
```

## 2.5 Zookeeper

The base version of ZooKeeper is Apache ZooKeeper 3.4.5.

ZooKeeper is a high-performance coordination service for distributed applications.

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

### 2.5.1 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-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm	
Type	Core
Requires	N/A

<b>Description</b>	Zookeeper core package which provides the executable, libraries, configuration files and documentations.
<b>Install on Nodes</b>	Every node in the ZooKeeper cluster, and the client workstations which will access the ZooKeeper service.

<b>zookeeper-server-&lt;PHD_ZOOKEEPER_VERSION&gt;-nn.noarch.rpm</b>	
<b>Type</b>	Deamon
<b>Requires</b>	ZooKeeper Core Package
<b>Description</b>	Daemon scripts package for Zookeeper server, which provides a convenient method to manage Zookeeper server start/stop as a Linux service.
<b>Install on Nodes</b>	N/A

<b>zookeeper-doc-&lt;PHD_ZOOKEEPER_VERSION&gt;-nn.noarch.rpm</b>	
<b>Type</b>	Documentation
<b>Requires</b>	N/A
<b>Description</b>	Zookeeper documentation.
<b>Install on Nodes</b>	N/A

## 2.5.2 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-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/zookeeper/rpm/zookeeper-server-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
```

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

## 2.5.3 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-<PHD_ZOOKEEPER_VERSION>-nn.noarch.rpm
```

## 2.5.4 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.

## 2.5.5 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:

## 2.6 HBase

The base version of HBase changed to Apache HBase 0.94.8.

HBase is a scalable, distributed database that supports structured data storage for large tables.

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

## 2.6.1 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.

## 2.6.2 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-<PHD_HBASE_VERSION>-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-<PHD_HBASE_VERSION>-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-<PHD_HBASE_VERSION>-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-<PHD_HBASE_VERSION>-nn.noarch.rpm	
Type	Daemon (thrift service)

<b>Requires</b>	HBase Core Package
<b>Description</b>	Daemon scripts package to provide HBase service through thrift.

<b>hbase-rest-&lt;PHD_HBASE_VERSION&gt;-nn.noarch.rpm</b>	
<b>Type</b>	Daemon (Restful service)
<b>Requires</b>	HBase Core Package
<b>Description</b>	Daemon scripts package to provide HBase service through REST.

<b>hbase-doc-&lt;PHD_HBASE_VERSION&gt;-nn.noarch.rpm</b>	
<b>Type</b>	Documentation
<b>Requires</b>	HBase Core Package
<b>Description</b>	HBase documentation.

## 2.6.3 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hbase/rpm/hbase-master-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

## 2.6.4 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hbase/rpm/hbase-regionserver-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

## 2.6.5 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
```



## 2.6.6 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hbase/rpm/hbase-thrift-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

## 2.6.7 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-<PHD_HBASE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hbase/rpm/hbase-rest-<PHD_HBASE_VERSION>-nn.noarch.rpm
```

## 2.6.8 HBase Configuration

The configuration files for HBase are located here: `/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.

## 2.6.9 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 `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
```

## 2.6.10 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
```

## 2.7 Hive

The base version of Hive is Apache Hive 0.11.0.

Hive is a data warehouse infrastructure that provides data summarization and ad hoc querying.

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

### 2.7.1 Hive Components

A Hive installation consists of the following components:

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

### 2.7.2 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:

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

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

## 2.7.3 Hive RPM Packages

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

hive-<PHD_HIVE_VERSION>-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-<PHD_HIVE_VERSION>-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-<PHD_HIVE_VERSION>-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

hive-server2-<PHD_HIVE_VERSION>-nn.noarch.rpm	
Type	Daemon (hive server2)
Requires	Hive Core Package
Description	Daemon scripts package to provide Hive Server2.

Install on Nodes	Hive Thrift server node
------------------	-------------------------

## 2.7.4 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 postgresql-jdbc
$ sudo rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hive/rpm/hive-metastore-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

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.war</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_2_0/scripts/metastore/upgrade/postgres/hive-schema-0.9.0.p
```

7. Start the hive-metastore:

```
$ service hive-metastore start
```

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

## 2.7.5 Hive Client Setup

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



```
$ sudo rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

## 2.7.6 Hive Thrift Server Setup

### [OPTIONAL]

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

```
$ sudo rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hive/rpm/hive-server-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

## 2.7.7 Hive Server2 Setup

### [OPTIONAL]

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

```
$ sudo rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hive/rpm/hive-server2-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

## 2.7.8 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 rpm -ivh working_dir/hive/rpm/hive-<PHD_HIVE_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hive/rpm/hive-metastore-<PHD_HIVE_VERSION>-nn.noarch.rpm
```

## 2.7.9 Hive Configuration

The configuration files for Hive are located here: `/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.

## 2.7.10 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
```

## 2.7.11 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 Beeline Client

HiveServer2 supports a new command shell Beeline that works with HiveServer2:

```
$ beeline
```

### 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 Server2

[Optional]

You can start/stop Hive server2 daemon as follows:

Run:

```
$ sudo service hive-server2 start
$ sudo service hive-server2 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 [Security](#) section to configure Hive for a security-enabled HD cluster.

## 2.8 Hcatalog

The base version of Hcatalog is Apache Hcatalog 0.11.0.

HCatalog is a metadata and table management system.

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

### 2.8.1 Prerequisites

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

## 2.8.2 Hcatalog RPM Packages

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

hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm	
<b>Type</b>	Core
<b>Requires</b>	Hadoop, HBase and Hive Core Packages.
<b>Description</b>	Hcatalog core package provides the executables, libraries, configuration files and documentations.
<b>Install on Nodes</b>	Hcatalog Client workstation.

hcatalog-server-<PHD_HCATALOG_VERSION>-nn.noarch.rpm	
<b>Type</b>	Daemon (hcatalog server).
<b>Requires</b>	Hcatalog Core Package.
<b>Description</b>	Daemon scripts package to provide Hive service through thrift.
<b>Install on Nodes</b>	Hcatalog server node.

webhcat-<PHD_HCATALOG_VERSION>-nn.noarch.rpm	
<b>Type</b>	Libraries.
<b>Requires</b>	Hcatalog Core Package.
<b>Description</b>	Daemon scripts package to provide Hive metadata information through metastore server.
<b>Install on Nodes</b>	Webhcat server node.

webhcat-server-<PHD_HCATALOG_VERSION>-nn.noarch.rpm	
<b>Type</b>	Daemon(webhcata server).
<b>Requires</b>	Hcatalog and Webhcat Core Package.
<b>Description</b>	Daemon scripts package to provide Webhcat Server.

<b>Install on Nodes</b>	Webhcat server node.
-------------------------	----------------------

## 2.8.3 Hcatalog Client Setup

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

```
$ sudo rpm -ivh working_dir/hive/rpm/hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
```

## 2.8.4 Hcatalog Server Setup

[OPTIONAL]

Install the Hcatalog core package and Hcatalog thrift daemon package to provide Hcatalog service.

```
$ sudo rpm -ivh working_dir/hcatalog/rpm/hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hcatalog/rpm/hcatalog-server-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
```

## 2.8.5 Webhcat Setup

[OPTIONAL]

Install the Hcatalog core package and Webhcat package to provide Webhcat libraries.

```
$ sudo rpm -ivh working_dir/hcatalog/rpm/hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hcatalog/rpm/webhcat-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
```

## 2.8.6 Webhcat Server Setup

[OPTIONAL]

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

```
$ sudo rpm -ivh working_dir/hcatalog/rpm/hcatalog-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hcatalog/rpm/webhcat-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/hcatalog/rpm/webhcat-server-<PHD_HCATALOG_VERSION>-nn.noarch.rpm
```

## 2.8.7 Hcatalog Configuration

The configuration files for Hcatalog are located here: `/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.

## 2.8.8 Usage

### Start Hcatalog Client

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

```
$ hcat
```

You can check the `hive` command usage by running:

```
$ hcat -help
```

### Start/Stop Hcatalog Server

You can start/stop Hcatalog server daemon as follows:

Either:

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

or:

```
$ sudo /etc/init.d/hcatalog-server start
$ sudo /etc/init.d/hcatalog-server stop
```

### Start/Stop Webhcat Server

You can start/stop Webhcat server daemon as follows:

Either:

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

or:

```
$ sudo /etc/init.d/webhcat-server start
$ sudo /etc/init.d/webhcat-server stop
```

## 2.9 Pig

The base version of Pig is Apache Pig 0.10.1.

Pig is a high-level data-flow language and execution framework for parallel computation.

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

### 2.9.1 Prerequisites

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

### 2.9.2 Pig RPM Packages

Pig has only one core package.

#### **pig-<PHD\_PIG\_VERSION>-nn.noarch.rpm**

<b>Type</b>	Core
<b>Requires</b>	Hadoop Core Packages
<b>Description</b>	Pig core package provides executable, libraries, configuration files and documentation.
<b>Install on Nodes</b>	Pig client workstation

#### **pig-doc-<PHD\_PIG\_VERSION>-nn.noarch.rpm**

<b>Type</b>	Documentation
<b>Requires</b>	N/A
<b>Description</b>	Pig documentation.
<b>Install on Nodes</b>	N/A

### 2.9.3 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-<PHD_PIG_VERSION>-nn.noarch.rpm
```

## 2.9.4 Pig Configuration

The configuration files for Pig are located here: `/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.

## 2.9.5 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
```

## 2.10 Mahout

The base version of Mahout is Apache Mahout 0.7.

Mahout is a scalable machine learning and data mining library.

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

### 2.10.1 Prerequisites

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

### 2.10.2 Mahout RPM Packages

Mahout has only one core package.

<b>mahout-&lt;PHD_MAHOUT_VERSION&gt;-nn.noarch.rpm</b>	
<b>Type</b>	Core
<b>Requires</b>	Hadoop Core Packages



<b>Description</b>	Mahout core package provides executable, libraries, configuration files and documentations.
<b>Install on Nodes</b>	Mahout client workstation

## 2.10.3 Mahout 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-<PHD_MAHOUT_VERSION>-nn.noarch.rpm
```

## 2.10.4 Mahout 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.

## 2.10.5 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
```

## 2.11 Flume

The base version of Flume is Apache Flume 1.3.1.

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.

## 2.11.1 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)

## 2.11.2 Flume RPM Packages

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

flume-<PHD_FLUME_VERSION>-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-<PHD_FLUME_VERSION>-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.

## 2.11.3 Flume Client Setup

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

```
$ sudo rpm -ivh working_dir/flume/rpm/flume-<PHD_FLUME_VERSION>-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

## 2.11.4 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-<PHD_FLUME_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/flume/rpm/flume-agent-<PHD_FLUME_VERSION>-nn.noarch.rpm
```

## 2.11.5 Flume Configuration

The configuration files for Flume are located here: `/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.

## 2.11.6 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

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
```

## 2.12 Sqoop

The base version of Sqoop is Apache Sqoop 1.4.2.

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.

### 2.12.1 Prerequisites

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

### 2.12.2 Sqoop RPM Packages

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

sqoop-<PHD_SQOOP_VERSION>-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-<PHD_SQOOP_VERSION>-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

### 2.12.3 Sqoop Client Setup

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

```
$ sudo rpm -ivh working_dir/sqoop/rpm/sqoop-<PHD_SQOOP_VERSION>-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.

## 2.12.4 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-<PHD_SQOOP_VERSION>-nn.noarch.rpm
$ sudo rpm -ivh working_dir/sqoop/rpm/sqoop-metastore-<PHD_SQOOP_VERSION>-nn.noarch.rpm
```

## 2.12.5 Sqoop Configuration

The configuration files for Flume are located here: /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.

## 2.12.6 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

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
```

## 3 Pivotal HD 1.1 Stack - Binary Package

### 3.1 Overview

Pivotal HD 1.1 is a full Apache Hadoop distribution with Pivotal add-ons and a native integration with the Pivotal Greenplum database.

PHD 1.1 Stack supports YARN (MR2) resource manager. You can submit Map-Reduce job via the new MapReduce interface.

The RPM distribution of PHD 1.1 contains the following:

- **HDFS 2.0.5-alpha**
- **Pig 0.10.1**
- **Zookeeper 3.4.5**
- **HBase 0.94.8**
- **Hive 0.11.0**
- **Hcatalog 0.11.0**
- **Mahout 0.7**
- **Flume 1.3.1**
- **Sqoop 1.4.2**

### 3.2 Accessing PHD 1.1 Stack Binary Package

You can download the PHD 1.1 Stack Binary Packages from EMC Download Center.

This is a single tar.gz file containing all the components: PHD-1.1.x.0-bin-xx.tar.gz. ( Here "x" denotes a digital number )

The content of this tar file looks like this:

```
PHD-1.1.0.0-bin-25/zookeeper/tar/zookeeper-3.4.5-gphd-2.1.0.0.tar.gz
PHD-1.1.0.0-bin-25/oozie/tar/oozie-3.3.2-gphd-2.1.0.0-distro.tar.gz
PHD-1.1.0.0-bin-25/hive/tar/hive-0.11.0-gphd-2.1.0.0.tar.gz
PHD-1.1.0.0-bin-25/flume/tar/apache-flume-1.3.1-gphd-2.1.0.0-bin.tar.gz
PHD-1.1.0.0-bin-25/pig/tar/pig-0.10.1-gphd-2.1.0.0.tar.gz
PHD-1.1.0.0-bin-25/hadoop/tar/hadoop-2.0.5-alpha-gphd-2.1.0.0.tar.gz
PHD-1.1.0.0-bin-25/mahout/tar/mahout-distribution-0.7-gphd-2.1.0.0.tar.gz
PHD-1.1.0.0-bin-25/sqoop/tar/sqoop-1.4.2-gphd-2.1.0.0-bin__hadoop-2.0.5-alpha-gphd-2.1.0.0.tar.gz
```

Note: md5 files are not listed here.

#### 3.2.1 Version Table

Here's the PHD version number for each components in this package:

0.11.0-gphd-2.1.0.0

Component	PHD Version	Version Placeholder
ZooKeeper	3.4.5-gphd-2.1.0.0	<PHD_ZOOKEEPER>
Hadoop	2.0.5-alpha-gphd-2.1.0.0	<PHD_HADOOP>
HBase	0.94.8-gphd-2.1.0.0	<PHD_HBASE>
Hive	0.11.0-gphd-2.1.0.0	<PHD_HIVE>
HCatalog	0.11.0-gphd-2.1.0.0	<PHD_HCATALOG>
Pig	0.10.1-gphd-2.1.0.0	<PHD_PIG>
Mahout	0.7-gphd-2.1.0.0	<PHD_MAHOUT>
Flume	1.3.1-gphd-2.1.0.0	<PHD_FLUME>
Sqoop	1.4.2-gphd-2.1.0.0.bin__hadoop-2.0.5-alpha-gphd-2.1.0.0	<PHD_SQOOP>
Oozie	3.3.2-gphd-2.1.0.0	<PHD_OOZIE>



In the sections below, we will use the values in the "Version Placeholder" to replace the actual PHD Version. When installing, please replace it back to the actual version value.

## 3.3 Installation

This section provides instructions for installing and running the Pivotal HD 1.1 components from the downloaded binary tarball files.



The installation instructions provided here are intended only as a Quick Start guide that will start the services on one single host. Refer to Apache Hadoop documentation for information about other installation configurations. <http://hadoop.apache.org/docs/r2.0.5-alpha/>



1. PHD should not be installed on the same cluster.
2. All packages used during this process should come from same PHD distribution tarball, do not mix using package from different tarballs.

### 3.3.1 Prerequisites

Follow the instructions below to install the Hadoop components (cluster install):

- If not created already, add a new user **hadoop** and switch to that user. All packages should be installed by user **hadoop** .

```
$ useradd hadoop
$ passwd hadoop
$ su - hadoop
```

- Make sure Oracle Java Run-time (JRE) 1.7 is installed on the system and set environment variable **JAVA\_HOME** to point to the directory where JRE is installed. Appending the following script snippet to the file **~/ .bashrc**:

```
~/ .bashrc

export JAVA_HOME=/usr/java/default
```

Make sure the **~/ .bashrc** file take effect:

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


- SSH (both client and server) command is required. Set up password-less SSH login according to the following commands.



Password-less SSH login is required to be setup on HDFS name node to each HDFS data node, also on YARN resource manager to each YARN node manager. Because we are setting up a single node cluster, which means the only machine is the HDFS name node, YARN resource manager, and the only HDFS data node YARN node manager. So the setup is easier.

```
# Assume you already log into the single node with user hadoop
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

# Set the permissions on the file on each slave host
$ chmod 0600 ~/.ssh/authorized_keys
```

 On a real cluster (distributed), use the following scripts, to setup password-less SSH login, it need to be executed twice, once on HDFS name node, another once on YARN resource manager node, unless you setup HDFS name node and YARN resource manager on same machine. (For your reference only, not needed for this single node cluster installation)

```
# First login to the master host (YARN resource manager or HDFS name node).
# Replace master@host-master with the real user name and host name of your master
host.
$ ssh master@host-master
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

# copy authorized_keys to each slave hosts (YARN node manager or HDFS data node)
in the cluster using scp
# Replace slave@host-slave with the real user name and host name of your slave
host, and do it for each of your slave host.
# NOTE: if an authorized_keys file already exists for# the user, rename your file
authorized_keys2
$ scp ~/.ssh/authorized_keys slave@host-slave:~/.ssh/

# Set the permissions on the file on each slave host
# Replace slave@host-slave with the real user name and host name of your slave
host, and do it for each of your slave host.
$ ssh slave@host-slave
$ chmod 0600 ~/.ssh/authorized_keys
```

## 3.3.2 Hadoop

1. Unpack the Hadoop tarball file

```
$ tar xzf hadoop-<PHD_HADOOP_VERSION>.tar.gz
```

2. Edit file `~/.bashrc` to update environment `HADOOP_HOME` and `HADOOP_HDFS_HOME` to be the directory where tarball file is extracted, and add `hadoop` to file search path.

```
~/.bashrc

# export HADOOP_HOME, HADOOP_HDFS_HOME
export HADOOP_HOME=/path/to/hadoop
```

```
export HADOOP_HDFS_HOME=/path/to/hadoop
export PATH=$HADOOP_HOME/bin:$PATH
```

3. And make sure the `~/.bashrc` file take effect:

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

4. In the sections below, all the shell commands, unless explicitly specified, are run from this `$HADOOP_HOME`.

## HDFS setup

1. Modify the file `$HADOOP_HOME/etc/hadoop/core-site.xml`, add the following to the configuration section

```
$HADOOP_HOME/etc/hadoop/core-site.xml

<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:8020</value>
</property>
```

2. Modify the file `$HADOOP_HOME/etc/hadoop/hdfs-site.xml`, add the following to the configuration section:

```
$HADOOP_HOME/etc/hadoop/hdfs-site.xml

<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
```

3. Format the HDFS name node directory using default configurations:

```
$ $HADOOP_HDFS_HOME/bin/hdfs namenode -format
```



The default location for storing the name node data is:

`/tmp/hadoop-hadoop/dfs/name/`

4. Start name node service:

```
$ $HADOOP_HDFS_HOME/sbin/hadoop-daemon.sh start namenode
```

5. Start each data node service:

```
$ $HADOOP_HDFS_HOME/sbin/hadoop-daemon.sh start datanode
```

6. After the name node and data node services are started, you can access the HDFS dashboard at <http://localhost:50070/>, if you are on using name node machine. If you use browser to open that dashboard from another machine, replace `localhost` in the URL with the full host name of your name node machine.

You can also do some test with the command line:


```
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -ls /  
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir -p /user/hadoop  
#you can see a full list of hdfs dfs command options  
$ $HADOOP_HDFS_HOME/bin/hdfs dfs  
#put a local file to hdfs  
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -copyFromLocal /etc/passwd /user/hadoop/
```

7. To stop data node service:

```
$ $HADOOP_HDFS_HOME/sbin/hadoop-daemon.sh stop datanode
```

8. To stop name node service:

```
$ $HADOOP_HDFS_HOME/sbin/hadoop-daemon.sh stop namenode
```

 HDFS data node and name node services are required to be started for running the examples below.

## YARN setup and run

### YARN setup

1. Modify the configuration file `$HADOOP_HOME/etc/hadoop/yarn-site.xml`, add the following to the configuration section:

```
$HADOOP_HOME/etc/hadoop/yarn-site.xml  
  
<property>  
  <name>yarn.nodemanager.aux-services</name>  
  <value>mapreduce.shuffle</value>  
</property>  
  
<property>  
  <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>  
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>  
</property>
```

```
<property>
  <description>Classpath for typical applications.</description>
  <name>yarn.application.classpath</name>

  <value>$HADOOP_CONF_DIR,$HADOOP_COMMON_HOME/share/hadoop/common/*,$HADOOP_COMMON_HOME/share/
</property>
```

2. Create basic directory on HDFS system for YARN usage:

```
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir /tmp
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -chmod -R 1777 /tmp
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir -p /user/hadoop
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir -p /user/history
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -chmod -R 1777 /user/history
```

3. Start YARN resource manager service:

```
$ $HADOOP_HOME/sbin/yarn-daemon.sh start resourcemanager
```

4. You can access the Resource Manager dashboard at: <http://localhost:8088/>, Replace the localhost in the URL for the full name of your resource manager host if you open that dashboard from another machine.

5. Start YARN node manager service:

```
$ $HADOOP_HOME/sbin/yarn-daemon.sh start nodemanager
```

6. You can access the node manager dashboard at: <http://localhost:8042/>. Replace the localhost in the URL for the full name of your node manager host if you open that dashboard from another machine.

At this time, you can do something with the YARN now.

7. If you want to stop YARN services now:

```
$ $HADOOP_HOME/sbin/yarn-daemon.sh stop nodemanager
$ $HADOOP_HOME/sbin/yarn-daemon.sh stop resourcemanager
```



YARN resource manager and node manager services are required to be started for running the following Map/Reduce examples.

## YARN to run Map/Reduce

1. Modify the file `$HADOOP_HOME/etc/hadoop/mapred-site.xml` (you may copy it from `$HADOOP_HOME/etc/hadoop/mapred-site.xml.template`), add the following to the configuration section:

```
$HADOOP_HOME/etc/hadoop/mapred-site.xml
```

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

2. Make sure YARN resource manager and node manager services are started (if not, refer to the "YARN Setup" step 2, 3).
3. If you want to track the mapreduce history, you can start the Map/Reduce history server service:

```
$ $HADOOP_HOME/sbin/mr-jobhistory-daemon.sh start historyserver
```

4. You can access the server dashboard at: <http://localhost:19888/>
5. Run Map/Reduce example:

```
$ cd $HADOOP_HOME
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-<PHD_HADOOP_VERSION>.jar
pi 2 10000
```

This command will submit the Map/Reduce job to calculate the PI value.

For more example, you can run:

```
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-<PHD_HADOOP_VERSION>.jar
```

It will show you a list of example programs you can run.

When the job is running, you can view the application progress at the resource manager dashboard.

### 3.3.3 Zookeeper

1. Unpack the Zookeeper tarball `zookeeper-<PHD_ZOOKEEPER_VERSION>.tar.gz` and add the `ZK_HOME` environment variable by appending the following to `~/.bashrc`:

```
~/.bashrc
```

```
# Add ZK_HOME to the path
export ZK_HOME=/path/to/zookeeper
PATH=$PATH:$ZK_HOME/bin
```

2. And make sure the `~/.bashrc` file take effect:

```
~/.bashrc
```

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

3. Go to the folder `$ZK_HOME/conf`,

```
$ cd $ZK_HOME/conf
$ cp zoo_sample.cfg zoo.cfg
```

Since you are running Zookeeper on a single node, no need to change the configuration file.

4. Start Zookeeper server service:

```
$ cd $ZK_HOME
$ bin/zkServer.sh start
```

5. Confirm that Zookeeper is running properly by running the following test:

```
$ cd $ZK_HOME
$ bin/zkCli.sh
> create /zk_test my_data
> get /zk_test
> quit
```

6. To stop the Zookeeper server service:

```
$ cd $ZK_HOME
$ bin/zkServer.sh stop
```

## 3.3.4 HBase

Following is an example of installing an instance of HBase that is running in pseudo-distributed mode. There is also an option to install a standalone or fully distributed HBase. Refer to Apache HBase documentation for information about other installation configurations. <http://hbase.apache.org/book/book.html>

1. Unpack the HBase tar file `hbase-<PHD_HBASE_VERSION>.tar.gz`, the extracted folder is referred as `$HBASE_HOME`, edit file `$HBASE_HOME/conf/hbase-site.xml` to add the following properties

**`$HBASE_HOME/conf/hbase-site.xml`**

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:8020/hbase</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
    <description>mode: fully distributed, not to manage zookeeper</description>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
```


```
<value>localhost</value>
</property>
</configuration>
```

2. Edit `$HBASE_HOME/conf/hbase-env.sh` to turn off the HBase management.

```
$HBASE_HOME/conf/hbase-env.sh
```

```
HBASE_MANAGES_ZK=false
```

3. HBase has the hadoop jars in the `$HBASE_HOME/lib` dir. If you already have `<PHD_HADOOP_VERSION>` version of Hadoop jar libraries in that directory, you can omit this step. Otherwise, you need:
  1. Delete the `$HBASE_HOME/lib/hadoop-*.jar` files.
  2. Copy the `$HADOOP_HOME/*/hadoop-*.jar` files to `$HBASE_HOME/lib/`.
4. Start HBase:

 Before starting HBase, please make sure Zookeeper server is running.

```
$ $HBASE_HOME/bin/start-hbase.sh
```

5. You can check the status of HBase at the following location: <http://localhost:60010>. If you using browser to open that dashboard from another machine, replace `localhost` in the URL with the full host name of your HBase master machine.
6. Confirm that HBase is installed and running properly by conducting the following test

```
$ cd $HBASE_HOME
$ bin/hbase shell
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> exit
```

7. To stop HBase:

```
$ $HBASE_HOME/bin/stop-hbase.sh
```



## 3.3.5 Hive

1. Unpack the Hive tarball `hive-<PHD_HIVE_VERSION>.tar.gz` and append the following environment variables to `~/ .bashrc`:

**~/ .bashrc**

```
export HIVE_HOME=/path/to/hive
export PATH=$HIVE_HOME/bin:$PATH
export CLASSPATH=$HIVE_HOME/lib:$CLASSPATH
```

2. Make sure the `~/ .bashrc` file take effect:

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

3. Create `/user/hive/warehouse` (aka `hive.metastore.warehouse.dir`) and set them group write access in HDFS

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir      /user/hive/warehouse
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w  /user/hive/warehouse
```

4. Test

```
$ cd $HIVE_HOME
$ bin/hive
hive> CREATE TABLE pokes (foo INT, bar STRING);
hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;
hive> SELECT a.* FROM pokes a where foo=400;
hive> DROP TABLE pokes;
hive> quit;
```

## 3.3.6 HCatalog

1. HCatalog is contained in the same tarball as Hive. After you extracted tarball `hive-<PHD_HIVE_VERSION>.tar.gz`, append the following environment variables to `~/ .bashrc`:

**~/ .bashrc**

```
export HCAT_HOME=$HIVE_HOME/hcatalog
export HCAT_PREFIX=$HCAT_HOME
export HIVE_CONF_DIR=$HIVE_HOME/conf
export HADOOP_LIBEXEC_DIR=$HADOOP_HOME/libexec
```

2. Make sure the `~/ .bashrc` file take effect:

**bash**

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

3. Now you can run some HCatalog commands to verify your setup is OK. You should see similar output as shown below. Some trivial output is omitted for better illustration:

**bash**

```
$ cd $HCAT_HOME
$ bin/hcat -e "create table pokes (foo int, bar string)"
OK
Time taken: 9.625 seconds
$ bin/hcat -e "show tables"
OK
pokes
Time taken: 7.783 seconds
$ bin/hcat -e "describe pokes"
OK
foo                int                None
bar                string              None
Time taken: 7.301 seconds
$ bin/hcat -e "alter table pokes add columns (new_col int)"
OK
Time taken: 7.003 seconds
$ bin/hcat -e "describe pokes"
OK
foo                int                None
bar                string              None
new_col            int                None
Time taken: 7.014 seconds
$ bin/hcat -e "drop table pokes"
OK
Time taken: 9.78 seconds
$ exit
```

## WebCatalog (Optional)

1. After you installed HCatalog, manually copy the configure file:

```
$ cp $HCAT_HOME/etc/webhcat/webhcat-default.xml $HIVE_CONF_DIR/webhcat-site.xml
```

2. Then edit the file you just copied:

**webhcat-site.xml**

```
<property>
  <name>templeton.exec.envs</name>
  <value>...,HIVE_CONF_DIR,HADOOP_LIBEXEC_DIR</value>
  <description>The environment variables passed through to exec.</description>
</property>
```

Please be noted the "..." in above script-let means the original value of the property. You need to append two more variable name to value of this property.

3. Start WebCatalog service:

```
$ cd $HCAT_HOME
$ sbin/webhcat_server.sh start
```

Note that starting WebCatalog service will write something under current directory, so ensure current user has permission to write in current directory.

4. Now you can run test:

```
$ curl http://localhost:50111/templeton/v1/ddl/database/?user.name=hadoop
```

5. Stop WebCatalog service:

```
bash

$ cd $HCAT_HOME
$ sbin/webhcat_server.sh stop
```

### 3.3.7 Pig

1. Unpack the Hive tarball `pig-<PHD_PIG_VERSION>.tar.gz` and append the following environment variables to `~/ .bashrc`:

```
export PIG_HOME=/path/to/pig
export PATH=$PIG_HOME/bin:$PATH
export CLASSPATH=$PIG_HOME/lib:$CLASSPATH
```

2. Make sure the `~/ .bashrc` file take effect:

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

3. Test

```
[hadoop@localhost ~]$ hadoop fs -put /etc/passwd passwd
[hadoop@localhost ~]$ pig
grunt> A = load 'passwd' using PigStorage(':');
grunt> B = foreach A generate $0 as id;
grunt> dump B;
(root)
(bin)
(daemon)
...
(flume)
```

```
(scoop)
(oozie)
grunt> quit;
```

The output in the above commands are omitted, after the **dump B** command, a Map/Reduce job should be started, and you should find users defined in your `/etc/passwd` file is listed in the output.

### 3.3.8 Mahout

1. Unpack the Mahout `mahout-distribution-<PHD_MAHOUT_VERSION>.tar.gz` and append the following environment variables to `~/ .bashrc`:

```
~/ .bashrc

export MAHOUT_HOME=/path/to/mahout
export PATH=$MAHOUT_HOME/bin:$PATH
export CLASSPATH=$MAHOUT_HOME/lib:$CLASSPATH
```

2. Make sure the `~/ .bashrc` file take effect:

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

3. Test (make sure HDFS and Map/Reduce service are running)

```
$ wget http://archive.ics.uci.edu/ml/databases/synthetic_control/synthetic_control.data
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir -p /user/hadoop/testdata
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -put synthetic_control.data testdata
$ $MAHOUT_HOME/bin/mahout org.apache.mahout.clustering.syntheticcontrol.kmeans.Job
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -ls -R output
```

### 3.3.9 Sqoop

1. [Install and Deploy MySQL](#)
2. Unpack the Sqoop `sqoop-<PHD_SQOOP_VERSION>.tar.gz` and append the following environment variables to `~/ .bashrc`:

```
~/ .bashrc

export SQOOP_HOME=/path/to/sqoop
export PATH=$SQOOP_HOME/bin:$PATH
export CLASSPATH=$SQOOP_HOME/lib:$CLASSPATH
```

3. Make sure the `~/ .bashrc` file take effect:

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

4. Move file `mysql-connector-java.jar` to directory `/usr/share/java/` and make a symbolic link point to it at sqoop's `lib` folder

```
$ ln -sf /usr/share/java/mysql-connector-java.jar $SQOOP_HOME/lib/mysql-connector-java.jar
```

5. Create user `hadoop` in MySQL system, and grant all privileges to the user.

```
$ mysql -u root [-p]

mysql> insert into mysql.user(Host,User>Password) values("%","hadoop",password("hadoop"));
mysql> GRANT ALL PRIVILEGES ON *.* TO 'hadoop'@'%' identified by 'hadoop';
mysql> flush privileges;
```

6. Start MySQL service

```
$ service mysqld start
```

7. Now do some test, first, create a table `student` in MySQL system:

```
$ mysql
mysql> use test
CREATE TABLE student (id INT PRIMARY KEY, name VARCHAR(100));
insert into student (id, name) values (1, "Elena");
insert into student (id, name) values (2, "Stephan");
insert into student (id, name) values (3, "Damon");
exit
```

8. Create a user home folder in HDFS, you are using user `hadoop`, create directory `/user/hadoop` in HDFS.
9. With user `hadoop` to execute:

```
[hadoop@localhost]$ sqoop import --connect jdbc:mysql://localhost/test --table student
--username hadoop --target-dir hdfs://localhost/tmp/sqoop_output"
```



If you installed MySQL on another machine, replace the `localhost` part in the jdbc url with the real MySQL server name in the command.

You should see a Map/Reduce job started to import data from the MySQL table to HDFS.

## 3.3.10 Flume

1. Unpack the Flume `apache-flume-<PHD_FLUME_VERSION>-bin.tar.gz` and append the following environment variables to `~/.bashrc`:

```
~/.bashrc
```

```
export FLUME_HOME=/path/to/flume
```

2. Make sure the `~/.bashrc` file take effect:

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

3. Create a Flume configuration file under `$FLUME_HOME` (assume you name it as `example.conf`), which you probably copy from `$FLUME_HOME/conf/flume-conf.properties.template`, according to the following example:

```
example.conf
```

```
# example.conf: A single-node Flume configuration
```

```
# Name the components on this agent
```

```
a1.sources = r1
```

```
a1.sinks = k1
```

```
a1.channels = c1
```

```
# Describe/configure the source
```

```
a1.sources.r1.type = netcat
```

```
a1.sources.r1.bind = localhost
```

```
a1.sources.r1.port = 44444
```

```
# Describe the sink
```

```
a1.sinks.k1.type = logger
```

```
# Use a channel which buffers events in memory
```

```
a1.channels.c1.type = memory
```

```
a1.channels.c1.capacity = 1000
```

```
a1.channels.c1.transactionCapacity = 100
```

```
# Bind the source and sink to the channel
```

```
a1.sources.r1.channels = c1
```

```
a1.sinks.k1.channel = c1
```

4. Run example use the example configuration to verify Flume is working properly.

```
$ cd $FLUME_HOME
```

```
$ bin/flume-ng agent --conf-file example.conf --name a1 -Dflume.root.logger=INFO,console
```

```
(note: on the above command, "a1" refers to the agent name set in file example.conf)
```

## 3.3.11 Oozie

1. Download and unpack Apache Tomcat 6.0.37 package
2. Download Ext JS 2.2 package from <http://extjs.com/deploy/ext-2.2.zip> and extract it to /tmp
3. Unpack the Oozie `oozie-<PHD_OOZIE_VERSION>-distro.tar.gz` and append the following environment variables to `~/ .bashrc`:


**~/ .bashrc**

```
export CATALINA_HOME=/path/to/tomcat
export OOZIE_HOME=/path/to/oozie
```

4. Make sure the `~/ .bashrc` file take effect:

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

5. Add the following configuration in Hadoop's configuration file `$HADOOP_HOME/etc/core-site.xml` and restart Hadoop.

 Replace `${username}` as your real user name which you use to start Oozie service (Probably user `hadoop`).

**core-site.xml**

```
<!-- OOZIE -->
<property>
  <name>hadoop.proxyuser.${username}.hosts</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.${username}.groups</name>
  <value>*</value>
</property>
```

6. Make a copy of Hadoop configuration file for Oozie

```
$ cp $HADOOP_HOME/etc/hadoop/* $OOZIE_HOME/conf/hadoop-conf
```

7. Initialize Oozie database

```
$ $OOZIE_HOME/bin/ooziedb.sh create -sqlfile oozie.sql -run
```

8. Setup Oozie

```
$ $OOZIE_HOME/bin/oozie-setup.sh prepare-war -hadoop 2.x $HADOOP_HOME -extjs /tmp/ext2.2
```


9. Setup Oozie share library. Replace `namenode-hostname` and `namenode-port` according to your Hadoop configurations

```
$ $OOZIE_HOME/bin/oozie-setup.sh sharelib create -fs
hdfs://${namenode-hostname}:${namenode-port} -locallib $OOZIE_HOME/oozie-sharelib.tar.gz
```

10. Start Oozie service

```
$ $OOZIE_HOME/bin/oozied.sh start
```

11. Run Oozie example.

 Replace `${username}` with your real user name when running the following commands

```
$ cd $OOZIE_HOME
$ tar xvf oozie-examples.tar.gz
$ sed -e 's/jobTracker=localhost:8021/jobTracker=localhost:8032/'
examples/apps/map-reduce/job.properties > temp; cp temp
examples/apps/map-reduce/job.properties
$ hdfs dfs -mkdir -p /user/${username}
$ hdfs dfs -put examples/ /user/${username}
$ ./bin/oozie job -oozie http://localhost:11000/oozie -config
examples/apps/ssh/job.properties -run
$ ./bin/oozie job -oozie http://localhost:11000/oozie -config
examples/apps/map-reduce/job.properties -run
```



## 4 Pivotal HD MR1 1.1 Stack - Binary Package

### 4.1 Overview

Pivotal HD MapReduce V1 (MR1) 1.1 is a full Apache Hadoop distribution with Pivotal add-ons and a native integration with the Pivotal Greenplum database.

The binary distribution of PHD MR1 1.1 contains the following:

- **HDFS 2.0.5-alpha**
- **MapReduce 1.0.3**
- **Pig 0.10.1**
- **Zookeeper 3.4.5**
- **HBase 0.94.8**
- **Hive 0.11.0**
- **Hcatalog 0.11.0**
- **Mahout 0.7**
- **Flume 1.3.1**
- **Sqoop 1.4.2**

### 4.2 Accessing PHD MR1 1.1

You can download the MR1 package PHDMR1-1.1.x.0-bin-xx.tar.gz from EMC Download Center, expand the package in your working\_dir:

```
$ tar zxvf PHDMR1-1.1.0.0-bin-xx.tar.gz
$ ls -l PHDMR1-1.1.0.0-bin-xx
total 44
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 flume
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hadoop
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hadoop-mr1
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hbase
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hive
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 hcatalog
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 mahout
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 pig
-rw-rw-r-- 1 hadoop hadoop 406 Jun 26 04:38 README
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 sqoop
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 utility
drwxr-xr-x 3 hadoop hadoop 4096 Jun 26 04:38 zookeeper
```

We define the replaced string which will be used in the following sections for each component.

Component	PHD Version	Replaced String
Hadoop	2.0.5_alpha_gphd_2_1_0_0	<PHD_HADOOP_VERSION>
MR1	mr1-1.0.3_gphd_2_1_0_0	<PHD_MR1_VERSION>
HBase	0.94.8_gphd_2_1_0_0	<PHD_HBASE_VERSION>
Hive	0.11.0_gphd_2_1_0_0	<PHD_HIVE_VERSION>
Pig	0.10.1_gphd_2_1_0_0	<PHD_PIG_VERSION>
Mahout	0.7_gphd_2_1_0_0	<PHD_MAHOUT_VERSION>
HCatalog	0.10.1_gphd_2_1_0_0	<PHD_HCATALOG_VERSION>
Sqoop	1.4.2_gphd_2_1_0_0	<PHD_SQOOP_VERSION>
Flume	1.3.1_gphd_2_1_0_0	<PHD_FLUME_VERSION>
Zookeeper	3.4.5_gphd_2_1_0_0	<PHD_ZOOKEEPER_VERSION>
Oozie	3.3.2_gphd_2_1_0_0	<PHD_OOZIE_VERSION>
bigtop-jsvc	1.0.15_gphd_2_1_0_0	<PHD_BIGTOP_J SVC_VERSION>
bigtop-utils	0.4.0_gphd_2_1_0_0	<PHD_BIGTOP_UTILS_VERSION>



- All component packages should come from same package (PHDMR1)

## 4.3 Installation

This section provides instructions for installing and running the Pivotal HD MR1 1.1.0 components from the downloaded binary tarball files.

The installation instructions provided here are intended only as a Quick Start guide that will start the services on one single host. Refer to Apache Hadoop documentation for information about other installation configurations. <http://hadoop.apache.org/docs/r2.0.5-alpha/>



1. PHDMR1 should not be installed on the same cluster.
2. All packages used during this process should come from same distribution tarball, do not mix using package from different tarballs.

## 4.3.1 Prerequisites

Follow the instructions below to install the Hadoop components (cluster install):

- If not created already, add a new user **hadoop** and switch to that user. All packages should be installed by user **hadoop**.

```
$ useradd hadoop
$ passwd hadoop
$ su - hadoop
```

- Make sure Oracle Java Run-time (JRE) 1.7 is installed on the system and set environment variable **JAVA\_HOME** to point to the directory where JRE is installed. Appending the following script snippet to the file `~/ .bashrc`:


```
~/ .bashrc

export JAVA_HOME=/usr/java/default
```

Make sure the `~/ .bashrc` file take effect:


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

- SSH (both client and server) command is required. Set up password-less SSH login according to the following commands.

 Password-less SSH login is required to be setup on HDFS name node to each HDFS data node, also on YARN resource manager to each YARN node manager. Because we are setting up a single node cluster, which means the only machine is the HDFS name node, YARN resource manager, and the only HDFS data node YARN node manager. So the setup is easier.

```
# Assume you already log into the single node with user hadoop
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

# Set the permissions on the file on each slave host
$ chmod 0600 ~/.ssh/authorized_keys
```

 On a real cluster (distributed), use the following scripts, to setup password-less SSH login, it need to be executed twice, once on HDFS name node, another once on YARN resource manager node, unless you setup HDFS name node and YARN resource manager on same machine. (For your reference only, not needed for this single node cluster installation)

```
# First login to the master host (YARN resource manager or HDFS name node).
# Replace master@host-master with the real user name and host name of your master
host.
$ ssh master@host-master
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

# copy authorized_keys to each slave hosts (YARN node manager or HDFS data node)
in the cluster using scp
# Replace slave@host-slave with the real user name and host name of your slave
host, and do it for each of your slave host.
# NOTE: if an authorized_keys file already exists for# the user, rename your file
authorized_keys2
$ scp ~/.ssh/authorized_keys slave@host-slave:~/.ssh/

# Set the permissions on the file on each slave host
# Replace slave@host-slave with the real user name and host name of your slave
host, and do it for each of your slave host.
$ ssh slave@host-slave
$ chmod 0600 ~/.ssh/authorized_keys
```

## 4.3.2 Hadoop

1. Unpack the Hadoop tarball file

```
$ tar xzf hadoop-<PHD_HADOOP_VERSION>.tar.gz
```

2. Edit file `~/.bashrc` to update environment `HADOOP_HOME` and `HADOOP_HDFS_HOME` to be the directory where tarball file is extracted, and add `hadoop` to file search path.

```
~/.bashrc

# export HADOOP_HOME, HADOOP_HDFS_HOME
export HADOOP_HOME=/path/to/hadoop

export HADOOP_HDFS_HOME=/path/to/hadoop
export PATH=$HADOOP_HOME/bin:$PATH
```

3. And make sure the `~/.bashrc` file take effect:

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

4. In the sections below, all the shell commands, unless explicitly specified, are run from this `$HADOOP_HOME`.

## HDFS setup

1. Modify the file `$HADOOP_HOME/etc/hadoop/core-site.xml`, add the following to the configuration section

```
$HADOOP_HOME/etc/hadoop/core-site.xml

<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:8020/</value>
</property>
```


2. Modify the file `$HADOOP_HOME/etc/hadoop/hdfs-site.xml`, add the following to the configuration section:

```
$HADOOP_HOME/etc/hadoop/hdfs-site.xml

<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
```

3. Format the HDFS name node directory using default configurations:

```
$ $HADOOP_HDFS_HOME/bin/hdfs namenode -format
```

 The default location for storing the name node data is:  
`/tmp/hadoop-hadoop/dfs/name/`

4. Start name node service:

```
$ $HADOOP_HDFS_HOME/sbin/hadoop-daemon.sh start namenode
```

5. Start each data node service:

```
$ $HADOOP_HDFS_HOME/sbin/hadoop-daemon.sh start datanode
```

6. After the name node and data node services are started, you can access the HDFS dashboard at <http://localhost:50070/>, if you are on using name node machine. If you are using browser to open that dashboard from another machine, replace `localhost` in the URL with the full host name of your name node machine.

You can also do some test with the command line:


```
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -ls /  
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir -p /user/hadoop  
#you can see a full list of hdfs dfs command options  
$ $HADOOP_HDFS_HOME/bin/hdfs dfs  
#put a local file to hdfs  
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -copyFromLocal /etc/passwd /user/hadoop/
```

#### 7. To stop data node service:

```
$ $HADOOP_HDFS_HOME/sbin/hadoop-daemon.sh stop datanode
```

#### 8. To stop name node service:


```
$ $HADOOP_HDFS_HOME/sbin/hadoop-daemon.sh stop namenode
```

 HDFS data node and name node services are required to be started for running the examples below.

## MapReduce v1 (MR1) Setup and Run

1. Unpack the MR1 tarball `hadoop-<PHD_MR1_VERSION>.tar.gz` and modify the `HADOOP_HOME` environment variable by appending the following to file `~/.bashrc`:

```
~/.bashrc  
  
# Add HADOOP_HOME to the path  
export HADOOP_HOME=/path/to/hadoop-mr1  
PATH=$HADOOP_HOME/bin:$PATH
```

 When you trying to start HDFS service, you need to specify the full path of `hadoop-daemon.sh` like `$HOME_HDFS_HOME/sbin/hadoop-daemon.sh`.

2. Edit the files under `hadoop-mr1/tar/hadoop-<PHD_MR1_VERSION>/conf/` directory and setup HDFS according to section **Prerequisite** and **HDFS Setup**.
3. Modify the file `$HADOOP_HOME/conf/mapred-site.xml`

```
conf/mapred-site.xml  
  
<configuration>  
  <property>  
    <name>mapred.job.tracker</name>  
    <value>localhost:8021</value>  
  </property>  
</property>
```

```
<name>mapred.system.dir</name>
<value>/mapred/system</value>
<final>true</final>
</property>
<property>
  <name>mapreduce.jobtracker.staging.root.dir</name>
  <value>/user</value>
</property>
</configuration>
```

4. Ensure you have already started HDFS services by checking you can access the HDFS dashboard: <http://localhost:50070/>, if you are on using name node machine. If you using browser to open that dashboard from another machine, replace `localhost` in the URL with the full host name of your name node machine.  
If you cannot access the dashboard, refer to **HDFS Setup** section to start the services.
5. Create basic directory on HDFS:

```
bash

$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir /tmp
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -chmod -R 1777 /tmp
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir -p /user/hadoop
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir -p /mapred/system
```

6. Start job tracker and task tracker services:

```
bash

$ $HADOOP_HOME/bin/hadoop-daemon.sh start jobtracker
$ $HADOOP_HOME/bin/hadoop-daemon.sh start tasktracker
```

7. Accessing map/reduce administration page at <http://localhost:50030>. If you using browser to open that dashboard from another machine, replace `localhost` in the URL with the full host name of your job tracker machine.
8. Now run an example Map/Reduce job to check MR1 is working

```
bash

$ cd $HADOOP_HOME
$ bin/hadoop jar hadoop-examples-*.jar pi 2 10000
<you should see job succeeded>
$ exit
```

9. If you can see the job is finished succeeded, you have setup PHDMR1 successfully, otherwise, check your configuration files and ensure all HDFS services, job tracker and task tracker all started successfully.

### 4.3.3 Zookeeper

1. Unpack the Zookeeper tarball `zookeeper-<PHD_ZOOKEEPER_VERSION>.tar.gz` and add the `ZK_HOME` environment variable by appending the following to `~/ .bashrc`:

```
~/bashrc

# Add ZK_HOME to the path
export ZK_HOME=/path/to/zookeeper
PATH=$PATH:$ZK_HOME/bin
```

2. And make sure the `~/ .bashrc` file take effect:

```
~/bashrc

$ source ~/ .bashrc
```

3. Go to the folder `$ZK_HOME/conf`,

```
$ cd $ZK_HOME/conf
$ cp zoo_sample.cfg zoo.cfg
```

Since you are running Zookeeper on a single node, no need to change the configuration file.

4. Start Zookeeper server service:

```
$ cd $ZK_HOME
$ bin/zkServer.sh start
```

5. Confirm that Zookeeper is running properly by running the following test:

```
$ cd $ZK_HOME
$ bin/zkCli.sh
> create /zk_test my_data
> get /zk_test
> quit
```

6. To stop the Zookeeper server service:

```
$ cd $ZK_HOME
$ bin/zkServer.sh stop
```

## 4.3.4 HBase

Following is an example of installing an instance of HBase that is running in pseudo-distributed mode. There is also an option to install a standalone or fully distributed HBase. Refer to Apache HBase documentation for information about other installation configurations. <http://hbase.apache.org/book/book.html>



1. Unpack the HBase tar file `hbase-<PHD_HBASE_VERSION>.tar.gz`, the extracted folder is referred as `$HBASE_HOME`, edit file `$HBASE_HOME/conf/hbase-site.xml` to add the following properties

**`$HBASE_HOME/conf/hbase-site.xml`**


```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:8020/hbase</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
    <description>mode: fully distributed, not to manage zookeeper</description>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>localhost</value>
  </property>
</configuration>
```

2. Edit `$HBASE_HOME/conf/hbase-env.sh` to turn off the HBase management.

**`$HBASE_HOME/conf/hbase-env.sh`**

```
HBASE_MANAGES_ZK=false
```

3. HBase has the hadoop jars in the `$HBASE_HOME/lib` dir. If you already have `<PHD_HADOOP_VERSION>` version of Hadoop jar libraries in that directory, you can omit this step. Otherwise, you need:
  1. Delete the `$HBASE_HOME/lib/hadoop-*.jar` files.
  2. Copy the `$HADOOP_HOME/*/hadoop-*.jar` files to `$HBASE_HOME/lib/`.
4. Start HBase:

 Before starting HBase, please make sure Zookeeper server is running.

```
$ $HBASE_HOME/bin/start-hbase.sh
```

5. You can check the status of HBase at the following location: <http://localhost:60010>. If you using browser to open that dashboard from another machine, replace `localhost` in the URL with the full host name of your HBase master machine.
6. Confirm that HBase is installed and running properly by conducting the following test

```
$ cd $HBASE_HOME
$ bin/hbase shell
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> exit
```

## 7. To stop HBase:

```
$ $HBASE_HOME/bin/stop-hbase.sh
```

## 4.3.5 Hive

1. Unpack the Hive tarball `hive-<PHD_HIVE_VERSION>.tar.gz` and append the following environment variables to `~/.bashrc`:

**~/.bashrc**

```
export HIVE_HOME=/path/to/hive
export PATH=$HIVE_HOME/bin:$PATH
export CLASSPATH=$HIVE_HOME/lib:$CLASSPATH
```

2. Make sure the `~/.bashrc` file take effect:

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

3. Create `/user/hive/warehouse` (aka `hive.metastore.warehouse.dir`) and set them group write access in HDFS

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir      /user/hive/warehouse
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w  /user/hive/warehouse
```

4. Test

```
$ cd $HIVE_HOME
$ bin/hive
hive> CREATE TABLE pokes (foo INT, bar STRING);
hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;
hive> SELECT a.* FROM pokes a where foo=400;
hive> DROP TABLE pokes;
hive> quit;
```

## 4.3.6 HCatalog

1. HCatalog is contained in the same tarball as Hive. After you extracted tarball

hive-<PHD\_HIVE\_VERSION>.tar.gz, append the following environment variables to ~/.bashrc:

**~/.bashrc**

```
export HCAT_HOME=$HIVE_HOME/hcatalog
export HCAT_PREFIX=$HCAT_HOME
export HIVE_CONF_DIR=$HIVE_HOME/conf
export HADOOP_LIBEXEC_DIR=$HADOOP_HOME/libexec
```

2. Make sure the ~/.bashrc file take effect:

**bash**

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

3. Now you can run some HCatalog commands to verify your setup is OK. You should see similar output as shown below. Some trivial output is omitted for better illustration:

**bash**

```
$ cd $HCAT_HOME
$ bin/hcat -e "create table pokes (foo int, bar string)"
OK
Time taken: 9.625 seconds
$ bin/hcat -e "show tables"
OK
pokes
Time taken: 7.783 seconds
$ bin/hcat -e "describe pokes"
OK
foo                int                None
bar                 string             None
Time taken: 7.301 seconds
$ bin/hcat -e "alter table pokes add columns (new_col int)"
OK
Time taken: 7.003 seconds
$ bin/hcat -e "describe pokes"
OK
foo                int                None
bar                 string             None
new_col             int                None
Time taken: 7.014 seconds
$ bin/hcat -e "drop table pokes"
OK
Time taken: 9.78 seconds
$ exit
```

## WebCatalog (Optional)

1. After you installed HCatalog, manually copy the configure file:

```
$ cp $HCAT_HOME/etc/webhcat/webhcat-default.xml $HIVE_CONF_DIR/webhcat-site.xml
```

2. Then edit the file you just copied:

**webhcat-site.xml**

```
<property>
  <name>templeton.exec.envs</name>
  <value>...,HIVE_CONF_DIR,HADOOP_LIBEXEC_DIR</value>
  <description>The environment variables passed through to exec.</description>
</property>
```

Please be noted the "." in above script-let means the original value of the property. You need to append two more variable name to value of this property.

3. Start WebCatalog service:

```
$ cd $HCAT_HOME
$ sbin/webhcat_server.sh start
```

Note that starting WebCatalog service will write something under current directory, so ensure current user has permission to write in current directory.

4. Now you can run test:

```
$ curl http://localhost:50111/templeton/v1/ddl/database/?user.name=hadoop
```

5. Stop WebCatalog service:

**bash**

```
$ cd $HCAT_HOME
$ sbin/webhcat_server.sh stop
```

## 4.3.7 Pig

1. Unpack the Hive tarball `pig-<PHD_PIG_VERSION>.tar.gz` and append the following environment variables to `~/ .bashrc`:

```
export PIG_HOME=/path/to/pig
export PATH=$PIG_HOME/bin:$PATH
export CLASSPATH=$PIG_HOME/lib:$CLASSPATH
```

2. Make sure the `~/ .bashrc` file take effect:

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

### 3. Test

```
[hadoop@localhost ~]$ hadoop fs -put /etc/passwd passwd
[hadoop@localhost ~]$ pig
grunt> A = load 'passwd' using PigStorage(':');
grunt> B = foreach A generate $0 as id;
grunt> dump B;
(root)
(bin)
(daemon)
...
(flume)
(sqoop)
(oozie)
grunt> quit;
```

The output in the above commands are omitted, after the **dump B** command, a Map/Reduce job should be started, and you should find users defined in your `/etc/passwd` file is listed in the output.

## 4.3.8 Mahout

1. Unpack the Mahout `mahout-distribution-<PHD_MAHOUT_VERSION>.tar.gz` and append the following environment variables to `~/.bashrc`:

```
~/.bashrc

export MAHOUT_HOME=/path/to/mahout
export PATH=$MAHOUT_HOME/bin:$PATH
export CLASSPATH=$MAHOUT_HOME/lib:$CLASSPATH
```

2. Make sure the `~/.bashrc` file take effect:

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

3. Test (make sure HDFS and Map/Reduce service are running)

```
$ wget http://archive.ics.uci.edu/ml/databases/synthetic_control/synthetic_control.data
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -mkdir -p /user/hadoop/testdata
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -put synthetic_control.data testdata
$ $MAHOUT_HOME/bin/mahout org.apache.mahout.clustering.syntheticcontrol.kmeans.Job
$ $HADOOP_HDFS_HOME/bin/hdfs dfs -ls -R output
```

## 4.3.9 Sqoop

### 1. [Install and Deploy MySQL](#)

### 2. Unpack the Sqoop

`sqoop-<PHD_SQOOP_VERSION>.bin_hadoop-<PHD_HADOOP_VERSION>.tar.gz` and append the following environment variables to `~/ .bashrc`:

```
~/ .bashrc
```

```
export SQOOP_HOME=/path/to/sqoop
export PATH=$SQOOP_HOME/bin:$PATH
export CLASSPATH=$SQOOP_HOME/lib:$CLASSPATH
```

### 3. Make sure the `~/ .bashrc` file take effect:

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

### 4. Move file `mysql-connector-java.jar` to directory `/usr/share/java/` and make a symbolic link point to it at sqoop's lib folder

```
$ ln -sf /usr/share/java/mysql-connector-java.jar $SQOOP_HOME/lib/mysql-connector-java.jar
```

### 5. Create user `hadoop` in MySQL system, and grant all privileges to the user.

```
$ mysql -u root [-p]

mysql> insert into mysql.user(Host,User>Password) values("","hadoop",password("hadoop"));
mysql> GRANT ALL PRIVILEGES ON *.* TO 'hadoop'@'%' identified by 'hadoop';
mysql> flush privileges;
```

### 6. Start MySQL service

```
$ service mysqld start
```

### 7. Now do some test, first, create a table `student` in MySQL system:

```
$ mysql
mysql> use test
CREATE TABLE student (id INT PRIMARY KEY, name VARCHAR(100));
insert into student (id, name) values (1, "Elena");
insert into student (id, name) values (2, "Stephan");
insert into student (id, name) values (3, "Damon");
exit
```

8. Create a user home folder in HDFS, you are using user `hadoop` , create directory `/user/hadoop` in HDFS.
9. With user `hadoop` to execute:

```
[hadoop@localhost]$ sqoop import --connect jdbc:mysql://localhost/test --table student
--username hadoop --target-dir hdfs://localhost/tmp/sqoop_output"
```



If you installed MySQL on another machine, replace the `localhost` part in the jdbc url with the real MySQL server name in the command.

You should see a Map/Reduce job started to import data from the MySQL table to HDFS.

## 4.3.10 Flume

1. Unpack the Mahout `apache-flume-<PHD_FLUME_VERSION>-bin.tar.gz` and append the following environment variables to `~/ .bashrc`:

**~/ .bashrc**

```
export FLUME_HOME=/path/to/flume
```

2. Make sure the `~/ .bashrc` file take effect:

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

3. Create a Flume configuration file under `$FLUME_HOME` (assume you name it as `example.conf`), which you probably copy from `$FLUME_HOME/conf/flume-conf.properties.template`, according to the following example:

**example.conf**

```
# example.conf: A single-node Flume configuration
```

```
# Name the components on this agent
```

```
a1.sources = r1
```

```
a1.sinks = k1
```

```
a1.channels = c1
```

```
# Describe/configure the source
```

```
a1.sources.r1.type = netcat
```

```
a1.sources.r1.bind = localhost
```

```
a1.sources.r1.port = 44444
```

```
# Describe the sink
```

```
a1.sinks.k1.type = logger
```

```
# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

4. Run example use the example configuration to verify Flume is working properly.

```
$ cd $FLUME_HOME
$ bin/flume-ng agent --conf-file example.conf --name a1 -Dflume.root.logger=INFO,console
(note: on the above command, "a1" refers to the agent name set in file example.conf)
```



## 5 Pivotal HD 1.1 Stack - Other Components

### 5.1 Overview

Pivotal HD 1.1 is a full Apache Hadoop distribution with Pivotal add-ons and a native integration with the Pivotal Greenplum database.

This section includes information about the following:

- [Spring Data](#)
- [HVE \(Hadoop Virtualization Extensions\)](#)
- [HDFS Rack Awareness](#)
- [Vaidya](#)
- [DataLoader](#)

For information about the installation, configuration, and use of the USS component is provided in the [USS Documentation](#) section.

### 5.2 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: [h <http://www.springsource.org/spring-data/hadoop>](http://www.springsource.org/spring-data/hadoop)

#### 5.2.1 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 PHDToolsversion.tar.gz
rw-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-- 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
```

## 5.2.2 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
```

## 5.2.3 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 the `docs/` directory to start using Spring Data Hadoop.

## 5.3 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`.

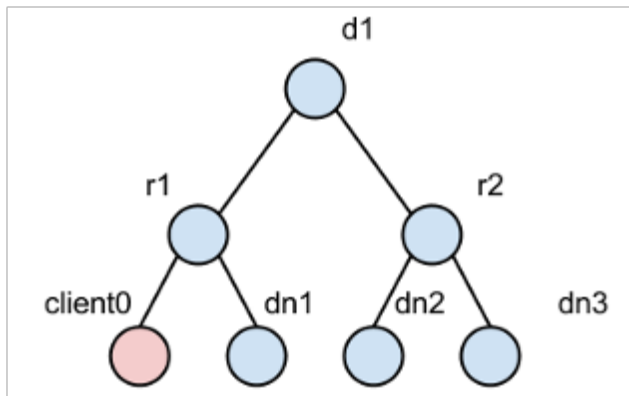
- Block pseudo distance sort for read locality: when reading a block, after obtaining all the located blocks, namenode sorts these located blocks based on their topological distance with client. The closer nodes get higher priority for read.

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.

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.

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

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

## 5.3.1 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.

## 5.4 Vaidya

### 5.4.1 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.

Note that Vaidya is not available if you are deploying a MR1-based cluster.

## 5.4.2 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/`

## 5.4.3 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</value>
</property>
```

### Disabling Vaidya

To disable Vaidya:

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

- 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.

## 5.4.4 Using Vaidya to Analyze Jobs

To use Vaidya with PHD 1.1 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 1.1 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**.

## 5.4.5 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.BalancedReducePartitioning]]></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-vaiddya-<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.vaiddya.DiagnosticTest` class:

**myrule.java**

```
package com.greenplum.vaiddya.rules;
import org.apache.hadoop.vaiddya.DiagnosticTest;
import org.apache.hadoop.vaiddya.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/g
</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.
- To make your settings take affect, restart job history server service.

## 5.5 DataLoader

See the *Pivotal DataLoader Installation and User Guide* for detailed information

## 6 USS Documentation - Main

### 6.1 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 metadata about mount points and delegated filesystems.

#### 6.1.1 Versions

This section contains information about USS version 0.5.0, which is compatible with Pivotal HD 1.0.1 and above (Hadoop version 2.0.x). USS 0.5.0 ships with Pivotal HD 1.1.

#### 6.1.2 Supported Features

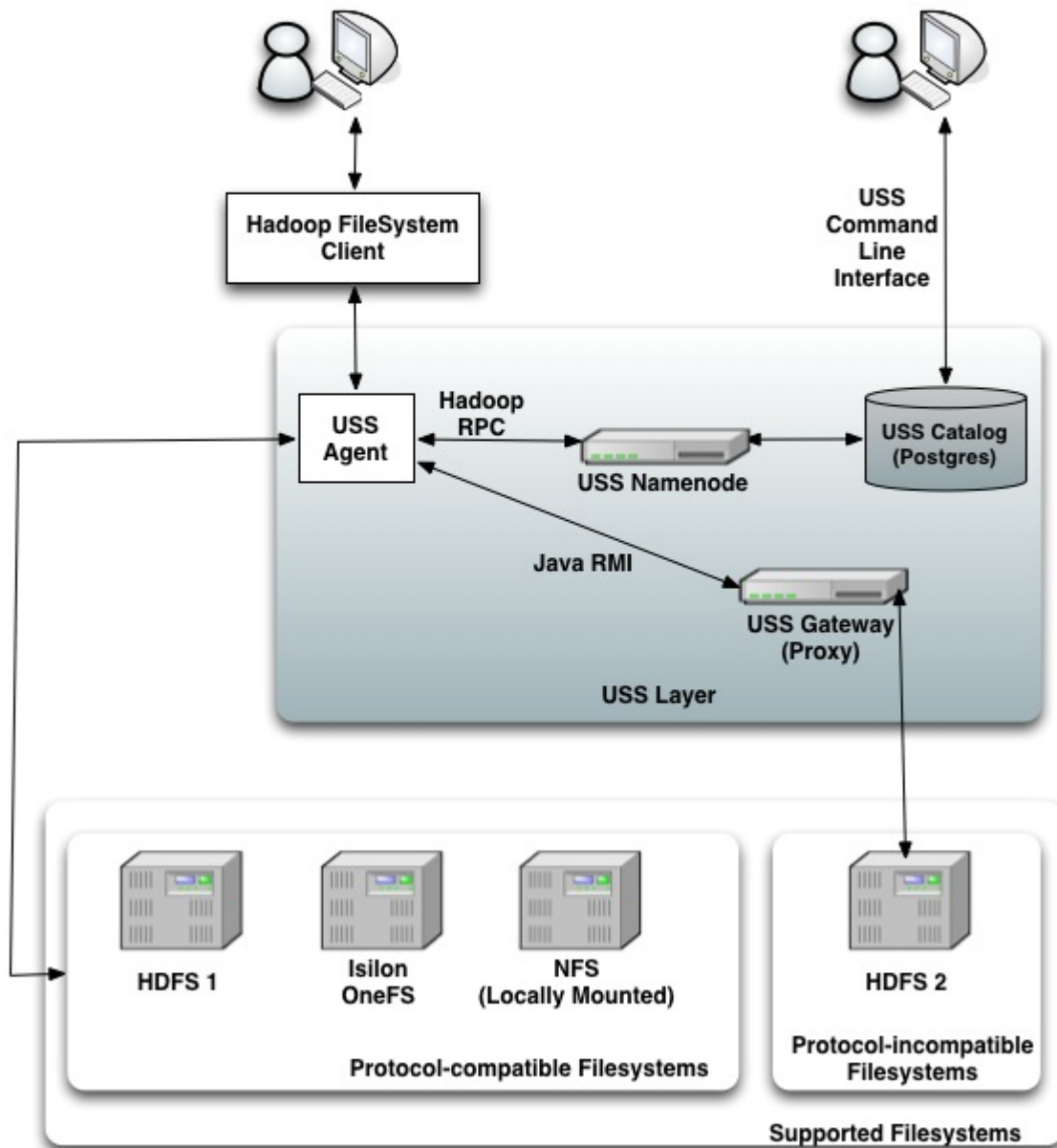
USS 0.5.0 supports the following features:

Feature	Support
Compatibility	Pivotal HD 1.0.1 and above (Hadoop 2.0.x)
Supported File Systems	<ul style="list-style-type: none"><li>• HDFS 2.x (Pivotal HD 1.0.1 and Pivotal HD 1.1)</li><li>• Cloudera CDH 4.4</li><li>• Isilon (OneFS)</li><li>• LocalFS (Mount point target paths assumed to be present on all datanodes/nodemanagers)</li><li>• NFS (Mounted on all datanodes/nodemanagers)</li><li>• FTP (with known issues)</li></ul>
Modes of access/interfaces	<ul style="list-style-type: none"><li>• HDFS (hadoop fs) CLI</li><li>• Native Java Map-Reduce Jobs</li><li>• Hadoop Streaming</li><li>• Pig</li></ul>

Feature	Support
	<ul style="list-style-type: none"><li>• Hive (External tables)</li></ul>
Installation/Configuration	Fully managed through Pivotal HD Manager 1.0.1 or above
Catalog	Postgres database
Security	With version 0.4.0, USS added support for secure HDFS clusters (using kerberos).
Protocol-incompatible Filesystem support	<p>With version 0.5.0, USS includes support for protocol incompatible Filesystems. Users can now access Filesystems with incompatible HDFS protocols through USS.</p> <p>As a first usecase for this feature, users can access (read and write) CDH4.4 clusters from Pivotal HD 1.1 clusters using USS.</p>

## 6.1.3 Architecture

## Unified Storage System



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

- Accepts Hadoop File System requests containing **USS URIs**
- 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](#) for a description of these terms.

### 6.1.4 Prerequisites

- Platform

RHEL 6.3/6.4 64Bit or CentOS 6.3/6.4 64Bit

- Pivotal HD Hadoop 1.0.1 or above

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 1.7 JDK (jdk7u15) needs to be installed in all USS nodes in the cluster. 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.

## 6.2 Getting Started

USS (v0.5.0) is part of Pivotal HD 1.1. USS is distributed as a set of three RPM packages that are part of the `PHDTools-version` tarball. You can download the PHDTools tarball from the [EMC Download Center](#) (paid version) or the [Pivotal HD product page](#) (Community Edition).

Architecturally, USS has the following four components:

### 6.2.1 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.

### 6.2.2 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.

## 6.2.3 USS Agent

The USS Agent is the FileSystem driver that is expected to be present on all Pivotal HD cluster nodes. This is the USS client-side library that 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.

## 6.2.4 USS Client

The USS Client hosts allow administrators to update the USS Catalog using the USS CLI. Pivotal HD Manager installs this role on all *Client Hosts* in your cluster configuration. Users can also use the USS Namenode or the USS Catalog host as the client node.

## 6.2.5 USS Gateway

USS uses a '*Gateway*' to enable access to protocol-incompatible filesystems. It is a host that acts as a bridge between a PHD cluster and a protocol-incompatible cluster by delegating the filesystem request to the target remote filesystem. This host is expected to have access to the wire-incompatible cluster that you wish to access using USS. The Pivotal HD Manager does not install this component, since it requires some information about the wire-incompatible cluster. More information about setting up this component can be found later in this document in the Setting up USS Gateway section.

### USS Directory Contents

The contents of USS 0.5.0 within the PHDTools-*version* tarball are located here:

```
$ tree PHDTools-1.1.0.0-90/uss/
PHDTools-1.1.0.0-90/uss/
  rpm
    uss-0.5.0-40.noarch.rpm
    uss-0.5.0-40.noarch.rpm.md5
    uss_catalog-0.5.0-40.noarch.rpm
    uss_catalog-0.5.0-40.noarch.rpm.md5
    uss_namenode-0.5.0-40.noarch.rpm
    uss_namenode-0.5.0-40.noarch.rpm.md5

1 directory, 6 files
```

RPM Package Name	Description	Dependencies	Hosts to install on
uss-0.5.0-40.noarch.rpm	This is the base USS package. It installs the USS library and	Pivotal HD hadoop package (version 2.0.2-alpha-gphd-2.0.1 and above)	All hosts using USS (agent, namenode, client and catalog hosts)

RPM Package Name	Description	Dependencies	Hosts to install on
	required configuration files.		
uss_catalog-0.5.0-40.noarch.rpm	This package installs the scripts and configuration files required to interact with and administer the USS Catalog.	uss (version >= 0.5.0)	USS Client hosts. Installed on the USS Namenode host by default as the <code>uss_namenode</code> RPM package has a dependency on it.
uss_namenode-0.5.0-40.noarch.rpm	This is the USS Namenode package. It installs the USS namenode service daemon and its runtime configuration.	uss_catalog package (version >= 0.5.0).	USS Namenode host

## 6.2.6 Package Contents

### uss-0.5.0-40.noarch.rpm

Files/Directories	Description
/usr/lib/gphd/uss/uss-0.5.0.jar	The USS library
/usr/lib/gphd/uss/uss-javadoc-0.5.0.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



Files/Directories	Description
/etc/gphd/uss/conf/uss-catalog.xml	USS Catalog configuration file
/etc/gphd/uss/conf/uss-env.sh	USS Client side environment setup script
/etc/gphd/uss/conf/uss.properties	USS Catalog database connection properties
/etc/gphd/uss/conf/uss-fs-metadata.xml	Sample Filesystem Registration configuration file for USS. This is a sample of the file that can be used to register a filesystem with USS using the USS CLI
/etc/default/uss	USS runtime configuration script. Exports environment variables that USS processes use.
/usr/lib/gphd/uss/libexec	Contains USS utility scripts.
/usr/lib/gphd/uss/libexec/uss-remote-proxy-launcher.sh	Script used to launch the USS Remote Proxy Server for accessing protocol-incompatible Filesystems (like Hadoop 1.x based HDFS)
/usr/lib/gphd/uss/libexec/uss-utils	Unix utilities for USS processes
/var/log/gphd/uss	USS log directory. Currently contains log files for the USS Remote Proxy Process
/var/run/gphd/uss	Directory for storing runtime files like pid files, lock files for USS processes
/usr/lib/gphd/uss/docs	The USS documentation directory. Contains README, Changelog, Notice and License files

## uss\_catalog-0.5.0-40.noarch.rpm

Files/Directories	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.

## uss\_namenode-0.5.0-40.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.
/var/log/gphd/uss-namenode	Log directory for USS Namenode

## 6.3 Installing USS

With Pivotal HD 1.0.1 and above 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.

## 6.4 Configuring USS

The following sections describe the USS configurations in Pivotal HD 1.0.1 and above.

 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`.

**/etc/gphd/hadoop/conf/core-site.xml**

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

```

Where:

Property	Description
fs.uss.impl	USS is implemented as a <i>pseudo</i> hadoop file system. USS URI's use the scheme <code>uss</code> . Hadoop compatible file systems set the property <code>fs.&lt;scheme&gt;.impl</code> to the class that implements the file system. The class that implements the USS <i>pseudo</i> file system is <code>com.greenplum.hd.uss.USSProtocolTranslator</code> .

## 6.4.1 USS Namenode Configuration

These are the server-side configurations for USS. The location of this configuration is `/etc/gphd/uss/conf/uss-nn-site.xml`.

**/etc/gphd/uss/conf/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>
  <property>
    <name>uss.remote.proxy.rmi.name</name>
    <value>RemoteFilesystem</value>
    <description>RMI Name for USS Remote Proxy</description>
  </property>
  <property>
    <name>uss.remote.proxy.timeout</name>
    <value>10000</value>
    <description>Timeout for USS Remote Proxy</description>
  </property>
  <property>
    <name>uss.remote.proxy.port</name>
    <value>1099</value>
    <description>Port that the remote proxy listens to</description>
  </property>

```

```

</property>
  <property>
    <name>uss.remote.proxy.launcher.script.path</name>
    <value>/usr/lib/gphd/uss/libexec/uss-remote-proxy-launcher.sh</value>
    <description>Path to the script to launch the USS Remote Proxy</description>
  </property>
  <property>
    <name>uss.remote.proxy.classpath.prefix</name>
    <value>/etc/gphd/uss/conf:/usr/lib/gphd/uss/*</value>
    <description>USS Classpath which pre-pended to the classpath of the remote proxy process
as provided by users while registering filesystems.</description>
  </property>
</configuration>

```

Where:

Property	Description
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.remote.proxy.rmi.name	Name of the RMI endpoint that the Remote Proxy Server on the USS Gateway starts up with. We do not recommend changing this property.
uss.remote.proxy.timeout	Timeout for the USS Remote Proxy startup.
uss.remote.proxy.port	Port that the remote proxy listens to. We do not recommend changing this property.
uss.remote.proxy.launcher.script.path	Path to the script to launch the USS Remote Proxy. We do not recommend changing this property.
uss.remote.proxy.classpath.prefix	USS Classpath which is pre-pended to the classpath of the remote proxy process as provided by users while registering filesystems. We do not recommend changing this property.

## 6.4.2 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:

Property	Description
fs.uss.name	This property specifies the hostname and port of the USS Namenode on the client side. By default, it is set to <code>uss://localhost:16040/</code> . 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 ( <code>.staging</code> directory) as well as to store the <code>.Trash</code> directory. As USS is implemented as a <i>pseudo</i> 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.

## 6.4.3 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_CATALOG_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.

See [Security](#) for information about USS Secure Configuration.

## 6.5 Using the USS Command Line Interface

USS provides a Command Line Interface for administrators to manage the USS catalog from the USS Client hosts. The purpose of this tool is to provide a central/consistent way to manage the USS Catalog. The USS Catalog resides in a postgres database and contains all the metadata related to registered filesystems and mount points.

### 6.5.1 Features

- The *USS Command Line Interface* is accessed via the command: `uss`. This is a script located at `/usr/bin/uss`
- All nodes part of the PHD Client role have this command available to them. It is a part of the `uss_catalog` RPM package. This utility is also available on the USS Namenode and the USS Catalog hosts (node on which the postgres database is installed).
- These nodes need to have network reachability to the USS Catalog host.

### 6.5.2 Usage

This section describes the usage of the USS Shell/CLI.

```
Usage: uss COMMAND
       where COMMAND is one of:
fs           run USS client commands to add/update filesystem metadata
mp           run USS client commands to add/update mount points
version      print the version
classpath    prints the class path used by USS commands
admin        run USS admin commands
```

The USS CLI, is divided to two levels of operations, one pertaining to Filesystem actions and one dealing with Mountpoint actions. Apart from these operations, the USS CLI also provides utilities that display version information and classpath for USS processes.

[Prepare Filesystem Configuration](#) -> [Register Filesystem](#) -> [Add Mountpoint\(s\)](#) -> [Verify Mountpoints](#)

USS Filesystem Operations

```
Usage: uss fs
[-list]
[-add <fsConfig>]
[-get <fsName>]
[-delete <fsName>]
[-help]
```

## Prepare Filesystem Configuration



You can find a sample configuration template at `/etc/gphd/uss/conf/uss-fs-metadata.xml`. You can copy this template to a custom directory and customize that based on your filesystem.

### Sample filesystem configuration

```
<configuration>
  <!-- Short string to describe the filesystem -->
  <property>
    <name>uss.fs.name</name>
    <value>PHD_FS1</value>
  </property>
  <!-- Scheme, hostname and port used to access the filesystem -->
  <property>
    <name>uss.fs.baseuri</name>
    <value>hdfs://NAMENODE_HOST1:8020</value>
  </property>
  <!-- Security type (simple or kerberos) -->
  <property>
    <name>uss.fs.security</name>
    <value>simple</value>
  </property>
  <!-- Primary Server Principal for Kerberos -->
  <property>
    <name>uss.fs.primaryServerPrincipal</name>
    <value></value>
  </property>
  <!-- Secondary Server Principal for Kerberos -->
  <property>
    <name>uss.fs.secondaryServerPrincipal</name>
    <value></value>
  </property>
  <!-- Filesystem access type (native or remote) -->
  <property>
    <name>uss.fs.access</name>
    <value>native</value>
  </property>
  <!-- Gateway hosts to access remote filesystems -->
  <property>
    <name>uss.fs.gatewayHosts</name>
    <value></value>
  </property>
  <!-- Classpath on the gatewayhosts to access remote filesystems -->
```

```
<property>
  <name>uss.fs.accessClasspath</name>
  <value></value>
</property>
</configuration>
```



If you wish to use a protocol incompatible filesystem, see section [Main-Protocol-IncompatibleFilesystemSupport](#)

If you wish to use a secure filesystem, see section [Main-SecureHDFSsupport](#)

## Add a Filesystem

The following command is to register a filesystem. Once the filesystem configuration is prepared, it is submitted to the USS catalog using the following command. If the filesystem name already exists in the catalog, a warning is shown.

Note : Please use the absolute path for the fsConfig.

```
Usage: uss fs -add <fsConfig>

$ uss fs -add uss-fs-metadata.xml
Filesystem 'PHD_FS1' registered successfully.
```

## List Filesystems

Lists all the registered filesystem names and base URIs.

```
$ uss fs -list
PHD_FS1 (1)    >    hdfs://NAMENODE_HOST1:8020
PHD_FS2 (2)    >    hdfs://NAMENODE_HOST2:8020
```

## List a Filesystem

Details of a particular filesystem can be listed using:

```
Usage: uss fs -list <fsName>

$ uss fs -list PHD_FS1
uss.fs.name - PHD_FS1
uss.fs.baseuri - hdfs://NAMENODE_HOST1:8020
uss.fs.security - simple
uss.fs.access - native
```

## Delete Filesystem



Filesystem can be deleted by providing a name to the following command.

Deleting a filesystem will delete **all** mount points associated with that filesystem.

```
Usage: uss fs -delete <fsName>

$ uss fs -delete PHD_FS1
```

## 6.5.3 USS Mountpoint Operations

```
Usage: uss mp
[-list]
[-add <mpName> <fsName> <fsPath>]
[-delete <mpName>]
[-help]
```

### Add a Mount Point

A mount point can be added to a registered filesystem using the following command. If the filesystem is not registered, the CLI errors out.

```
Usage: uss mp -add <mpName> <fsName> <fsPath>

$ uss mp -add click_data PHD_FS1 /data/clicks
```

### List Mount Points

List command now outputs the mount point name, the filesystem it sits on and the path on that filesystem. The change is that, the base URI is not very prominent as before.

```
$ uss mp -list
click_data (1)    >   hdfs://NAMENODE_HOST1:8020/data/click
revenue_data (2)  >   hdfs://NAMENODE_HOST2:8020/data/revenue

$ uss mp -list click_data
click_data (1)    >   hdfs://NAMENODE_HOST1:8020/data/click
```

### Delete a Mount Point


Deleting a mount point will not delete its filesystem.


```
Usage: uss mp -delete <mpName>

$ uss mp -delete click_data
```

## 6.5.4 Initialize USS Catalog

- Creates the usscatalog database on the host which has the postgres installation to use as the USS Catalog Database
- Creates the usscatalog database owner
- Creates the usscatalog database DDL
- Adds default data to the database

 Pivotal HD Manager initializes the USS catalog database using this command automatically during cluster deployment.

 USS uses [flyway](#) to manage schema versioning and migration.

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

## 6.5.5 USS Classpath

Utility to view the java classpath used by USS.

```
$ uss classpath
```

## 6.5.6 USS Version

Displays version information for the installed USS build.

```
$ uss version
USS Version 0.5.0
SCM ssh://git@stash.greenplum.com:2222/phd/uss.git -r f5e51910fe816290fa6434b31a424ebb79c724ca
Compiled by jenkins on 2013-10-18T01:06Z
From source with checksum dec405d789c2d68d6a2f6ed16139
```

## 6.6 Protocol-Incompatible Filesystem Support

In an environment where multiple Hadoop filesystems (of differing Hadoop versions) coexist, all filesystems might not be accessible via Pivotal HD's Hadoop client. This is because the protocol versions of the Hadoop Client and HDFS might not always be compatible.

## 6.6.1 Supported Protocol-Incompatible Filesystem

USS version 0.5.0 supports access to Cloudera's CDH 4.4 distribution (Hadoop 2.0.0-cdh4.4.0).

### Gateway Setup

The [Gateway node](#) needs certain libraries and configurations from the protocol-incompatible filesystems. These need to be copied to a staging directory on the Gateway host. The staging directory needs to be specified in the `uss.fs.accessClasspath` (colon separated) during filesystem registration into the USS Catalog.

Detailed steps to do this are as follows:

- Locate the hadoop client libraries of the protocol-incompatible filesystem. For Cloudera's CDH 4.4, they are usually found in a location like `/opt/cloudera/parcels/CDH/lib/hadoop/client` on all CDH cluster nodes.
- We recommend that you copy the libraries to `/home/uss/gateway/client` directory of the Gateway Node.

```
scp CDH_NODE:/opt/cloudera/parcels/CDH/lib/hadoop/client/*
GATEWAY_NODE:/home/uss/gateway/client
```

- Ensure all libraries in `/home/uss/gateway/client` have read permission for user 'uss'.

```
$ ls -ltrh /home/uss/gateway/client
-rwxr-xr-x 1 uss uss 1.3M Oct 12 22:50 zookeeper-3.4.5-cdh4.4.0.jar
-rwxr-xr-x 1 uss uss 4.4M Oct 12 22:50 hadoop-hdfs-2.0.0-cdh4.4.0.jar
...
```

- Locate the client configurations of the protocol-incompatible filesystem. For Cloudera's CDH 4.4, they are usually found in a location like `/etc/hadoop/conf` of the target filesystem.
- We recommend that you copy the libraries to `/tmp/uss/gateway/conf` directory of the Gateway.

```
scp CDH_NODE:/etc/hadoop/conf/* GATEWAY_NODE:/home/uss/gateway/conf
```

- Ensure all the configuration files in `/home/uss/gateway/conf` have read permission for user 'uss'.

```
$ ls -ltrh /home/uss/gateway/conf
-rw-r--r-- 1 uss uss 1.1K Oct 12 22:50 core-site.xml
-rw-r--r-- 1 uss uss 1.5K Oct 12 22:50 hdfs-site.xml
...
```

- Ensure that the `fs.defaultFS` in `/home/uss/gateway/conf/core-site.xml` points to a reachable IP address or hostname of the target filesystem's namenode.

Note: This command can be used to test reachability

```
nc -z <CDH_NAMENODE> <CDH_NAMENODE_PORT>
```

- Currently only a single Gateway host is supported for a registered filesystem

## 6.6.2 Passphraseless Access Setup

The Gateway process is started by the USS NameNode. This requires passphrase-less SSH access from the USS NameNode Host to the Gateway Host (for user 'uss').

- Firstly, login as user 'uss' into the USS NameNode.

```
$ sudo -u uss
```

- Create your public/private keys:

```
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

- Copy the public key from the USS NameNode to the ~/.ssh/authorized\_keys of Gateway Node

```
cat ~/.ssh/id_dsa.pub | ssh uss@<GATEWAY_HOST> "cat >> .ssh/authorized_keys"
```

- Note
  - Ensure the permissions of ~/.ssh directory is 700
  - Ensure the permissions of ~/.ssh/authorized\_keys file is 640
- To confirm password-less access, try to ssh to Gateway from the USSNameNode and you should **not** be prompted for a password.

## 6.6.3 Register a Protocol-Incompatible Filesystem

A protocol-incompatible filesystem is registered via the USS CLI. Details about registering filesystems in USS are available in the section [Main-PrepareFilesystemConfiguration](#)

Note: Make a copy of /etc/gphd/uss/conf/uss-fs-metadata.xml before editing the original.

The 4 important pieces of metadata about the protocol-incompatible filesystem are:

1. **Base URI:** This is specified in the `uss.fs.baseuri` element of the `uss-fs-metadata.xml`. This field identifies the filesystem URI of the protocol-incompatible HDFS. The URI usually is

"hdfs://<NAMENODE\_HOST>:<NAMENODE\_PORT>/"

. Ensure that the NAMENODE\_HOST is reachable from the Gateway node. This is especially important if the protocol-incompatible is on a different backplane.

2. **Filesystem Access Type:** This is specified in the `uss.fs.access` element of the `uss-fs-metadata.xml`. Setting `uss.fs.access` to **'remote'** tells USS that this is a protocol-incompatible filesystem. Its default value is `native` for protocol-compatible Filesystems.
3. **Gateway Host:** This is specified in the `uss.fs.gatewayHosts` element of the `uss-fs-metadata.xml`. This element specifies a single hostname or IP address of the Pivotal HD node which is designated as a Gateway to access the protocol-incompatible filesystem.
  1. Note: Ensure that the hostname or the IP used in this field is reachable from the Pivotal Hadoop Clients.
  2. Only **one** host can be specified as the gateway
4. **Access Classpath:** This is specified in the `uss.fs.accessClasspath` element of the `uss-fs-metadata.xml`. Based on the recommendation in the GatewaySetup section, the `uss.fs.accessClasspath` element would look like the following. Note: Notice the asterisk after the client directory (`/home/uss/gateway/client/*`) and none after the conf directory (`/home/uss/gateway/conf`).

#### Sample `uss-fs-metadata.xml` for protocol-incompatible filesystems

```
<configuration>
  <!-- Short string to describe the filesystem -->
  <property>
    <name>uss.fs.name</name>
    <value>CDH_4.4_Filesystem</value>
  </property>
  <!-- Scheme, hostname and port used to access the filesystem -->
  <property>
    <name>uss.fs.baseuri</name>
    <value>hdfs://CDH_NAMENODE:8020</value>
  </property>
  <!-- Security type (simple or kerberos) -->
  <property>
    <name>uss.fs.security</name>
    <value>simple</value>
  </property>
  <!-- Primary Server Principal for Kerberos -->
  <property>
    <name>uss.fs.primaryServerPrincipal</name>
    <value></value>
  </property>
  <!-- Secondary Server Principal for Kerberos -->
  <property>
    <name>uss.fs.secondaryServerPrincipal</name>
    <value></value>
  </property>
  <!-- Filesystem access type (native or remote) -->
  <property>
    <name>uss.fs.access</name>
    <value>remote</value>
  </property>
  <!-- Gateway hosts to access remote filesystems -->
```

```
<property>
  <name>uss.fs.gatewayHosts</name>
  <value>GATEWAY_HOSTNAME</value>
</property>
<!-- Classpath on the gatewayhosts to access remote filesystems -->
<property>
  <name>uss.fs.accessClasspath</name>
  <value>/home/uss/gateway/conf:/home/uss/gateway/client/*</value>
</property>
</configuration>
```

## 6.6.4 Adding a Mount Point

Once the filesystem is registered successfully in the USS Catalog, add a USS Mount Point on that filesystem as usual (`uss mp -add <mountpoint> <filesystem> <directory>`). See section [Add a mount point](#).

## 6.6.5 Testing Access

After the above steps are complete, the protocol-incompatible filesystem should be accessible via Hadoop Command Line, Java Map Reduce jobs, Hive, Pig...etc. Refer to [Testing USS](#) section for more example. A mount point on a protocol-incompatible filesystem can be accessed just like a native mountpoint.

A simple test would be to list the contents of the mount point:

```
$ hadoop fs -ls uss://<USS_NN_HOST>:<USS_NN_PORT>/<MOUNTPOINT>
```

## 6.7 Secure HDFS support

To access a secure HDFS cluster through USS, add the security configuration for the cluster to the USS catalog while registering the filesystem with USS using the USS Command Line Interface (CLI) described earlier in this document.

### 6.7.1 Secure HDFS and MountPoint Registration

Secure HDFS is registered via the USS CLI by providing three additional important pieces of metadata about the secure HDFS cluster:

- **Security Type:** This is specified in the `uss.fs.security` element of the `uss-fs-metadata.xml`. Setting `uss.fs.security` to **'kerberos'** tells USS that this is a secure HDFS cluster. Its default value is `simple` for non-secure Filesystems.
- **Primary Server Principal:** This is specified in the `uss.fs.primaryServerPrincipal` element of the `uss-fs-metadata.xml`. This element specifies the `kerberos` principal of the secure HDFS cluster's Namenode. Its value is specified by the property `dfs.namenode.kerberos.principal` in the `hdfs-site.xml` of the secure cluster.

- **Secondary Server Principal:** This is specified in the `uss.fs.secondaryServerPrincipal` element of the `uss-fs-metadata.xml`. This element specifies the kerberos principal of the secure HDFS cluster's Secondary Namenode. Its value is specified by the property `dfs.secondary.namenode.kerberos.principal` in the `hdfs-site.xml` of the secure cluster.

## 6.8 Using USS

### 6.8.1 Setting up the environment

Once USS is configured, you 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:

#### USS Environment Settings

```
$ export HADOOP_CLASSPATH=/usr/lib/gphd/uss/uss-0.5.0.jar:/etc/gphd/uss/conf/
$ export HADOOP_USER_CLASSPATH_FIRST=true
```

### Using USS Paths in map-reduce jobs

If you wish to use USS paths in map-reduce jobs, you need to make one more configuration step. When the nodemanagers try to access USS paths, they need the USS library to resolve these paths. As a result, you 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-0.5.0.jar:/etc/gphd/uss/conf/
export HADOOP_USER_CLASSPATH_FIRST=true
```

The yarn configuration file also needs to be updated, as follows:

In `/etc/gphd/hadoop/conf/yarn-site.xml` append the following string to the `yarn.application.classpath` property's value:

```
| /usr/lib/gphd/uss/*:/etc/gphd/uss/conf
```

### 6.8.2 Testing USS

To test USS, you 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

### Hadoop CLI examples with USS

```
# 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

### Java Map Reduce example with USS

```
$ hadoop jar /usr/lib/gphd/hadoop-mapreduce/hadoop-mapreduce-examples.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 example with USS

```
\# 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-0.5.0.jar uss.pig // Execute pig script, by  
adding the uss jar as an additional jar to include.
```

## Examples using USS URIs in Hive Scripts

### Hive example with USS

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

## Examples using USS URIs in Streaming jobs

### Hadoop Streaming example with USS

```
\# 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.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

```

## Accessing USS URIs in Java programs, Map-Reduce jobs

Users can access the Filesystem in Java programs or Map Reduce jobs. To use USS URIs in such applications, please use the following API to access the FileSystem -

```
FileSystem fs = FileSystem.get("uss://USS_NAMENODE:16040/", conf);
```



The alternative `FileSystem.get(conf)` API provides access to the default FileSystem (most likely HDFS). USS URIs do not work with this FileSystem handles obtained using this API.

## 6.9 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`.

### 6.9.1 Starting the Service

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

### 6.9.2 Stopping the Service

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

### 6.9.3 Restarting the Service

```
service uss-namenode restart
```

### 6.9.4 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:

File	Description
<code>/var/run/uss-namenode.pid</code>	USS Namenode service pid file.
<code>/var/lock/subsys/uss-namenode</code>	USS Namenode service lock file.
<code>/var/log//gphd/uss-namenode/uss-namenode.log</code>	USS Namenode log file

## 6.10 Uninstallation

We recommend using Pivotal HD Manager to uninstall USS from your cluster.

You can also uninstall USS by erasing the USS RPM packages from the hosts on which it was installed.

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

## 6.11 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.

### Unexpected SQL error occurred

Please check if the USS Catalog is running and initialized correctly.

**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
$ 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 for 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/conf/hadoop-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.java: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, 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?

```
hadoop fs -ls '/'
```

```
$ 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
```

```
uss mp -list
```

```
$ uss mp -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 example, 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](#). 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-0.5.0.jar <pig_script_using_uss_paths>
```

**Why do I get an error relation "mount\_point" does not exist when I try to add a mount-point using `uss mp -add`?**

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`.

**Failed on local exception: `java.net.SocketException: Host is down...` (or) Failed on connection exception: `java.net.ConnectException: Connection refused`**

`uss.fs.baseuri` configured in the filesystem configuration Namenode/datanode is not reachable, or is not started.

**Remote proxy server failed to start**

`uss.fs.baseuri` configured incorrectly in the filesystem configuration

**Remote proxy server failed to start - ssh: Could not resolve hostname localhost: nodename nor servname provided, or not known**

`uss.fs.gatewayHosts` configured incorrectly in filesystem configuration. Only one host can be configured here. Also make sure that the USS Namenode has passphraseless access to the Gateway Node, see section [Main-PassphraselessAccessSetup](#)

**Wrong FS: `hdfs://NAMENODE:8020`, expected: `hdfs://NAMENODE_OTHER:8020`**

Value of *fs.defaultFS* in *core-site.xml* on the gateway node's client side staging directory is not consistent with *uss.fs.baseuri* configured in the filesystem configuration

Also make sure the same gateway node is not used for accessing another remote filesystem

**Wrong FS:** `hdfs://NAMENODE:8020/FILEPATH`, **expected:** `file:///`

Missing remote filesystem client configuration in the staging directory on the gateway node. Make sure that you have included the `conf/` directory in the *uss.fs.accessClasspath* in filesystem configuration.

Also make sure the directory exists on the gateway host

**Remote proxy server failed to start - Exception in thread "main" java.lang.NoClassDefFoundError:**

Missing remote filesystem client library jars in the staging directory on the gateway node. Make sure that you have included the client library directory in the *uss.fs.accessClasspath* in filesystem configuration.

Also make sure the directory exists on the gateway host along with all the client jar's of the remote filesystem

**Failed on local exception: com.google.protobuf.InvalidProtocolBufferException: Message missing required fields**

Make sure the client jar's in the staging directory on the gateway host is compatible with the remote cluster.

## 6.12 Terminology

Term	Description
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



Term	Description
	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.
Protocol-compatible FileSystem	For a Pivotal HD HDFS cluster say cluster1, a protocol-compatible FileSystem is one with which cluster1's HDFS clients can communicate over Hadoop RPC. Multiple versions of HDFS may be protocol-compatible with each other. Any such cluster's HDFS Client can communicate with any other such HDFS cluster. In USS, to access such a filesystem, users do not need to configure a Gateway.
Protocol-incompatible FileSystem	For a Pivotal HD HDFS cluster say cluster1, a protocol-incompatible FileSystem is one with which cluster1's HDFS clients can not communicate over Hadoop RPC. This is because cluster1's Hadoop RPC protocol is incompatible with the other filesystem Hadoop RPC protocol version. In USS, to access such a filesystem, users need to configure a Gateway.
Gateway	A Gateway host on a Pivotal HD cluster enables users to access a protocol-incompatible HDFS cluster using USS. A Gateway is essentially a pseudo-client for the target protocol-incompatible filesystem. It runs with the context (configuration, libraries) of the protocol-incompatible filesystem and helps resolve protocol-incompatibility.
Filesystem Configuration	Every registered filesystem has a corresponding filesystem configuration that contains metadata about the filesystem.

## 7 HVE (Hadoop Virtualization Extensions)

### 7.1 Overview

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.

### 7.2 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.

#### 7.2.1 Topology Awareness Configuration and Verification

##### Sample Setup

This setup has 2 logical racks, 2 physical hosts (install ESXi and managed by vCenter) per rack, and 2 DN/CN (VM in ESXi) nodes per host. There is also one NameNode/Jobtracker and a client node that can be used to start jobs.

In this setup, each DN/CN node has 4 vCPUs, 16G memory and 200G (Non-SSD) disks.

The NameNode and JobTracker are installed on another dedicated VM with 4vCPU, 4G Memory and 100G disks.

Node Distribution on Hosts:

<b>Rack 1</b>	<b>Host 1</b>	<b>NameNode and JobTracker</b>	<b>DN1</b>
	<b>Host 2</b>	<b>DN2</b>	<b>NN3</b>
<b>Rack 2</b>	<b>Host 3</b>	<b>DN4</b>	<b>DN5</b>
	<b>Host 4</b>	<b>DN6</b>	<b>DN7</b>

## Enable topology awareness (Hadoop V2)

1. Add the following line to `core-site.xml`:

```
<property>
  <name>topology.script.file.name</name>
  <value>/hadoop/hadoop-smoke/etc/hadoop/topology.sh</value>
```

```
[root@namenode enable]# cat topology.data
10.111.57.223(VM IP)    /Rack1/NodeGroup1
10.111.57.224          /Rack1/NodeGroup1
10.111.57.225          /Rack1/NodeGroup2
10.111.57.226          /Rack2/NodeGroup1
10.111.57.227          /Rack2/NodeGroup1
10.111.57.228          /Rack2/NodeGroup2
10.111.57.229          /Rack2/NodeGroup2
```

### Topology.sh sample:

```
[root@namenode enable]# cat topology.sh
#!/bin/bash
HADOOP_CONF=/hadoop/hadoop-smoke/etc/hadoop
# this is the location of topology.data
while [ $# -gt 0 ] ; do
    nodeArg=$1
    exec< ${HADOOP_CONF}/topology.data
    result=""
    while read line ; do
        ar=( $line )
        if [ "${ar[0]}" = "$nodeArg" ] ; then
            result="${ar[1]}"
        fi
    done
    shift
    if [ -z "$result" ] ; then
        echo -n "/default/rack "
    else
        echo -n "$result "
    fi
done
```

### 3. Verify HVE is enabled:Run the TestDFSIO script:

The output is as follows:

```
1)HVE enabled:
Job Counters
Launched map tasks=100
Launched reduce tasks=1
Data-local map tasks=26
NODEGROUP_LOCAL_MAPS=49
Rack-local map tasks=25
2)HVE disabled:
Job Counters
Launched map tasks=100
Launched reduce tasks=1
Data-local map tasks=20
Rack-local map tasks=80
```

## 7.3 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.

### 7.3.1 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.

### 7.3.2 Function List

Below are functionalities included in this elastic feature:

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.

### 7.3.3 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>
```

## 7.3.4 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 <hostname> <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: `${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:

Hostname	Role
NameNode	JobTracker / NameNode
Hadoop-01	TaskTracker / DataNode
Hadoop-02	TaskTracker / DataNode

### 1. CLI

#### 1. 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
```

#### 2. 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
}

```

### 3. 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

```

### 4. 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

```

## 5. 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

```

## 6. setSlot (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

### 1. 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

```

## 2. 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

```

## 8 Security

---

### 8.1 Security

---

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

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

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 \(MapReduce\)](#)
- [Zookeeper Secure Configuration](#)
- [HBase Secure Configuration](#)
- [Hive Secure Configuration](#)
- [MapReduce Version 1 Configuration \(MRv1\)](#)
- [Auditing](#)
- [Secure Web Access](#)
- [Troubleshooting](#)

### 8.2 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 be configured.

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

#### 8.2.1 Kerberos Set-up

## Install the KDC

If you do not have a pre-existing KDC see [Installing the MIT Kerberos 5 KDC](#).

**Note:** CentOS and RedHat use AES-256 as the default encryption strength. If you want to use AES-256 you will 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.

### 8.2.2 Install Kerberos Workstation and Libraries on Cluster Hosts

If you are using MIT krb5 run:

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

### 8.2.3 Distribute the Kerberos Client Configuration File to all Cluster Hosts

If you are using Kerberos 5 MIT that 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.

### 8.2.4 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 fully qualified hostname ( `host_fqdn`); one you could use to connect to the host in question.

**Important:**

- Replace `REALM` with the KDC realm you are using for your PHD cluster where it appears.
- The host names used **MUST** be resolvable to an address on all the cluster hosts and **MUST** be of the form `host.domain` as some Hadoop components require at least one "." part in the host names used for principals.
- The names of the principals seem to matter as some processes may throw exceptions if you change them. Hence it is safest to use the specified Hadoop principal names.
- Hadoop supports an `_HOST` tag in the site XML that is interpreted as the `host_fqdn` but this must be used properly. See [Using \\_HOST in Site XML](#).

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

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

- For each cluster host (excepting client-only hosts) run:

```
addprinc -randkey HTTP/host_fqdn@REALM
```

- HDFS (name node, data nodes), for each HDFS service host run:

```
addprinc -randkey hdfs/host_fqdn@REALM
```

- YARN (resource manager, node managers), for each YARN service host run:

```
addprinc -randkey yarn/host_fqdn@REALM
```

- MAPRED (job history server): for each JHS service host run:

```
addprinc -randkey mapred/host_fqdn@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.

## 8.2.5 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 have the service principal for that host and the HTTP principal for that host in the file.

### 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/host_fqdn@REALM HTTP/host_fqdn@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 three node cluster (one node name node, two data nodes):

```
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/host_fqdn@REALM HTTP/hostname@REALM
```

For example, for a three node cluster (one node resource manager; two 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/host_fqdn@REALM HTTP/host_fqdn@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
```

## 8.2.6 Distribute the Keytab Files

1. On each cluster node create the directory for the keytab files; here 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:
  1. Change the permissions on all key tabs to read-write by owner only:  
`chmod 400 *.keytab`
  2. Change the group on all keytab files to `hadoop`:  
`chgrp hadoop *`
  3. 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`
  4. 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.

This 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
```

This 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
```

## 8.2.7 Java Support Items Installation

### Install JCE on all Cluster Hosts

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

**Note:** These files will already exist in your environment and look 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 the following:
3. Install JSVC using `yum` and run the check again; if it fails try the next step.
4. Build from source and install manually (see [Building and Installing JSVC](#)).

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 step 2 above. JSVC is very picky about platform and JDK matching, so use the [Building and Installing JSVC](#) instructions for your system OS and JDK.

## 8.2.8 Container and Script Modifications

### Configure the Linux Container

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

```
# 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:

```
---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:

```
-rw-r--r-- 1 root root 363 Jul 4 00:29
/usr/lib/gphd/hadoop-yarn/etc/hadoop/container-executor.cfg
```

If they do not then set them as follows:

```
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 data nodes

1. Uncomment the lines at the bottom 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 however you built or hand-installed JSVC your `JSVC_HOME` will be `/usr/bin` so you must set it appropriately, so modify `/etc/default/hadoop` and set 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.

## 8.2.9 Site XML Changes

### Using `_HOST` in Site XML

You may maintain consistent site XML by using the `_HOST` keyword for the `host_fqdn` part in the site XML if:

- Your cluster nodes were identified with fully qualified domain names when configuring the cluster.
- `hostname -f` on all nodes yields the proper fully qualified hostname (same as the one used when creating the principals).

You cannot use constructs like `_HOST.domain`; these will be interpreted literally. You can only use `_HOST` in the site XML, files such as `jaas.conf` needed for Zookeeper and HBase must use actual FQDN's for hosts.

### Edit the Site XML

Finally we are ready to edit the site XML to turn on secure mode. Before getting into this it is good to understand who needs to talk to whom. By "talk" we mean using 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 log in 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 data node needs to talk to the name node
- Each node manager needs to talk to the resource manager and the job history server
- Each client/gateway node needs to talk to the name node, resource manager, and job history server

**Important:**

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

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

1. Edit `/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>

<!-- THE PROPERTY BELOW IS OPTIONAL: IT ENABLES ON WIRE RPC ENCRYPTION -->

<property>
  <name>hadoop.rpc.protection</NAME>
  <value>privacy</value>
</property>
```

2. Edit `/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>

<!-- short circuit reads do not work when security is enabled -->

<property>
  <name>dfs.client.read.shortcircuit</name>
  <value>false</value>
```

```
</property>
<!-- name node 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-host_fqdn@REALM</value>
</property>

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

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

<!-- (optional) secondary name node 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-host_fqdn@REALM</value>
</property>

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

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

<!-- data node 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 data nodes -->

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

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

<!-- 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-host_fqdn@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>

<!-- THE PROPERTIES BELOW ARE OPTIONAL AND REQUIRE RPC PRIVACY (core-site): THEY ENABLE ON
WIRE HDFS BLOCK ENCRYPTION -->

<property>
  <name>dfs.encrypt.data.transfer</name>
  <value>true</value>
</property>

<property>
  <name>dfs.encrypt.data.transfer.algorithm</name>
```

```
<value>rc4</value>
<description>may be "rc4" or "3des" - 3des has a significant performance
impact</description>
</property>
```

3. Edit `/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-host_fqdn@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-host_fqdn@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.LinuxContainerExecutor</value>
</property>

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

4. Edit `/usr/lib/gphd/hadoop/etc/hadoop/mapred-site.xml` as follows:

```
<!-- 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-host_fqdn@REALM</value>
</property>
```

## 8.2.10 Complete the HDFS/YARN Secure Configuration

1. Start the cluster:

```
icm_client start
```

2. Check that all the processes start up. If not go to the appendix on troubleshooting.
  - Control processes: namenode, resourcemanager, historyserver should all be running.
  - Cluster worker processes: datanode and namenode should be running.

**Note:** Until you do 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 go to the [Troubleshooting](#) section.

7. [optional] Set the sticky bit on the `/tmp` directory (prevents non-super users from moving or deleting other users files in `/tmp`):
  1. Login as `gpadmin` on any HDFS service node (namenode, datanode)
  2. Execute the following:

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

3. Execute the following:

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



4. 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 see the [Troubleshooting](#) section.

## 8.2.11 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 data nodes.
3. Either:
  1. If you made backups as suggested above:  
Restore the original site xml files  
or:
  2. If you do not have backups, then edit the site xml as follows:
    - Set Linux container executable to `org.apache.hadoop.yarn.server.nodemanager.DefaultContainerExecuto:` on all data nodes
    - Set `dfs.block.access.token.enable` to `false` on all data nodes
    - 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

## 8.2.12 Building and Installing JSVC

In order for the data nodes to start as root to get secure ports and then switch back to the `hdfs` user `jsvc` must be installed ([http://commons.apache.org/proper/commons-daemon/download\\_daemon.cgi](http://commons.apache.org/proper/commons-daemon/download_daemon.cgi)). If the packaged `jsvc` binary is not working we recommend building `jsvc` from source for your platform.

You only need to do 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](http://commons.apache.org/proper/commons-daemon/download_daemon.cgi)
3. scp it to one of your data node 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 run these exports before 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 found by running:

```
which jsvc
```

The correct output is:

```
/usr/bin/jsvc
```

10. Run:

```
jsvc -help
```

Look under the printed `-jvm` item and you should see something like:

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

If the line under Available JVMs (where server is above) is blank there is a problem as it cannot find the JVM; check that the JDK is installed properly in `/usr/java/default`.

## 8.2.13 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* by 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 just use the `EXAMPLE.COM` REALM if you want.
  - 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_encetypes` 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_encetypes = 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

This typically runs quickly but may take 5-10 minutes if the code has trouble getting the random bytes it needs

#### 5. Add an administrator account as `username/admin@REALM`

Run the `kadmin.local` application from the command line `kadmin.local: addprinc username/admin@REALM`

Type `quit` to exit `kadmin.local`

**Important:** The KDC does not need to be running to add a principal.

#### 6. Start the KDC by running:

`/etc/init.d/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:

```
kadmin.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 `kadmin` cannot (such as use the `-norandkey` option in `ktadd`)

## 9. Start kadmind:

```
/etc/init.d/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:
  1. Login as the admin you created: `kinit username/admin`
  2. 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 should now have a KDC that is functional for PHD secure cluster operations.

## 8.3 Zookeeper Secure Configuration

Zookeeper secure configuration for server is recommended for HBase.

**Important:** STOP cluster services before doing this configuration.

### 8.3.1 Zookeeper Servers

#### Create the Zookeeper Principals

Create a principal for each Zookeeper Quorum Server host:

```
kadmin: addprinc -randkey zookeeper/host_fqdn@REALM
```

#### Create the Zookeeper Keytab Files

For each Zookeeper server host:

```
ktadd -norandkey -k /etc/security/keytab/zookeeper-hostid.service.keytab  
zookeeper/host_fqdn@REALM
```

#### Distribute the Zookeeper Keytab Files

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 zookeeper 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.SASLAuthenticationProvider
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.keytab"
    storeKey=true
    useTicketCache=false
    principal="zookeeper/host_fqdn@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/zookeeper/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](#).
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 testznode data sasl:zkcli@REALM:cdwra
```

#### 4. Verify the znode:

```
getAcl /testznode:
```

You should see something like this:

```
'sasl,'zkcli@{BIGDATA.COM%7D%7D  
: cdrwa
```

## 8.3.2 Zookeeper Clients

Optional.

#### 1. Add a principal for the client on the client host:

```
kadmin.local: addprinc -randkey zclient/host_fqdn@REALM
```

#### 2. Add the keytab:

```
kadmin.local: ktadd -norandkey -k /etc/security/keytab/zclient-hostid.client.keytab  
zclient/host_fqdn@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/host_fqdn@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/zookeeper/conf/jaas.conf  
-Dsun.net.spi.nameservice.provider.1=dns,sun"
```

**Important:**

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 starting `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.

## 8.4 HBase Secure Configuration

If you are running secure HBase you should also also run a secure Zookeeper (see [Zookeeper Configuration](#) above). 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>).

### 8.4.1 HBase Master and RegionServers

#### Create the HBase Principals

For the HBase master and each region server host run:

```
kadmin.local: addprinc -randkey hbase/host_fqdn@REALM
```

Where `host_fqdn` refers to the service principal (master, regionserver) host.

#### Create the HBase Keytab files

For the HBase master and each region server host run:



```
kadmin.local: ktadd -norandkey -k /etc/security/keytab/hbase-hostid.service.keytab
hbase/host_fqdn@REALM
```

## Distribute the HBase Keytab Files

For each host:

Move the appropriate keytab file for each host to that hosts `/etc/security/keytab` directory, then run:

```
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

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.TokenProvider</value>
</property>

<!-- HBase secure region server configuration -->
<property>
  <name>hbase.regionserver.kerberos.principal</name>
  <value>hbase/regionserver-host_fqdn@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-host_fqdn@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 `/etc/init.d/hbase-master start` or `/etc/init.d/hbase-regionserver start` to check that the issue is resolved.

## 8.4.2 HBase Clients

Add 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, to do so add 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 here: <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 `ac` table in HBase. Adding the permissions below are sufficient per HBase documentation:

```
grant 'rest-principal-name', 'RWCA'
```

## Thrift Client Configuration

See the HBase documentation here: <http://hbase.apache.org/book/security.html> for instructions on configuring Thrift clients.

### 8.4.3 HBase with Secure Zookeeper Configuration

For secure HBase you should also run a secure Zookeeper (see [Zookeeper 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.

1. Create a file `/etc/gphd/hbase/conf/jaas.conf` and the following:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  useTicketCache=false
  keyTab="/etc/security/keytab/hbase.service.keytab"
  principal="hbase/host_fqdn@REALM";
};
```

**Important:** Make sure to replace `host_fqdn@REALM` with the `host_fqdn` of the server and the correct `REALM`.

2. Add the following near at the bottom 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:

```
<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:

```
kerberos.removeHostFromPrincipal=true
kerberos.removeRealmFromPrincipal=true
```

## 8.5 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>
```

## 8.6 USS Secure Configuration

USS adds support for secure Hadoop clusters. To work on a secure HDFS cluster, users have to add additional metadata about their cluster's security parameters to the USS catalog. In version 0.5.0, the metadata can be added using the USS CLI described later in this document.

See [USS documentation](#) for information about Accessing A secure HDFS Cluster through USS.

### 8.6.1 Securing USS

To configure security in USS using Kerberos, add the following properties to the USS configuration files. After doing so, restart the USS namenode.

#### uss-client-site.xml and uss-nn-site.xml

Property	Description
<code>uss.security.authentication</code>	This property indicates the authentication that USS must use. It is set to <code>kerberos</code> for secure hadoop clusters. For non-secure clusters, this property defaults to <code>simple</code> .
<code>uss.namenode.principal</code>	This property is set to the kerberos principal of the USS Namenode. The secure tools can generate the principal and set this property.

## 8.7 MapReduce Version 1 Configuration (MRv1)

To configure MRv1 follow the same steps as for HDFS and YARN except for the following differences:

### 8.7.1 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 run: `addprinc -randkey mapred/host_fqdn@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.

## 8.7.2 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
```

## 8.7.3 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-host_fqdn@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-host_fqdn@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</value>
</property>

<property>
  <name>mapreduce.tasktracker.group</name>
  <value>mapred</value>
</property>
```

## 8.8 Auditing

You can enable auditing before deployment or re-configuration of a cluster.

To enable auditing:

1. Locate your templates directory (by default `ClusterConfigDir`, this is created during initial installation, see *Pivotal HD Enterprise 1.1 Installation and Administrator Guide* for details).
2. For HDFS and MapReduce, locate the `hdfs` subdirectory and edit the `log4j.properties` file as follows:

For HDFS change line:

```
hdfs.audit.logger=INFO,NullAppender
```

to:

```
hdfs.audit.logger=INFO,RFAAUDIT
```

For MapReduce change line:

```
mapred.audit.logger=INFO,NullAppender
```

to:

```
mapred.audit.logger=INFO,RFAAUDIT
```

For other components, locate the component sub-directory in the template and its corresponding `log4j.properties` file and make similar edits.

To specify auditing output location:

By default, log files and other auditing information is output to `/var/log/gphd/hadoop/`.

To set up logging to go to syslog, define the following:

```
# Configure syslog appender
log4j.appender.SYSLOG=org.apache.log4j.net.SyslogAppender
log4j.appender.SYSLOG.syslogHost=loghost
log4j.appender.SYSLOG.layout=org.apache.log4j.PatternLayout
log4j.appender.SYSLOG.layout.ConversionPattern=%d{ISO8601} %p %c: %m%n
log4j.appender.SYSLOG.Facility=LOCAL1
```

You can now log audit information to syslog, for example:

```
hdfs.audit.logger=INFO,SYSLOG
```

You can also log to file and syslog, for example:

```
hdfs.audit.logger=INFO,RFAAUDIT,SYSLOG
```

Note that these changes only go into effect after deployment or re-configuration.

## 8.9 Secure Web Access

This section describes how to configure WebHDFS on a secure PHD cluster.

### 8.9.1 Overview

WebHDFS is a REST API that allows a user to perform various HDFS operations.

More details about these APIs are here: [WebHDFS REST API](#)

On an insecure cluster, the user can run any `webhdfs` command as any user, including as `root`, for example:

```
$ curl -i "http://<HOST>:<PORT>/webhdfs/v1/?user.name=root&op=LISTSTATUS"
```



**Note:** Where `<HOST>:<PORT>` is the HTTP address of the namenode (the value of `dfs.http.address` in the `hdfs-site.xml`). By default, the port number is 50070.

The client receives a JSON response that looks like this:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 427

{
  "FileStatuses":
  {
    "FileStatus":
    [
      {
        "accessTime"      : 1320171722771,
        "blockSize"       : 33554432,
        "group"           : "supergroup",
        "length"          : 24930,
        "modificationTime": 1320171722771,
        "owner"           : "webuser",
        "pathSuffix"       : "a.patch",
        "permission"       : "644",
        "replication"      : 1,
        "type"             : "FILE"
      },
      {
        "accessTime"      : 0,
        "blockSize"       : 0,
        "group"           : "supergroup",
        "length"          : 0,
        "modificationTime": 1320895981256,
        "owner"           : "szetszwo",
        "pathSuffix"       : "bar",
        "permission"       : "711",
        "replication"      : 0,
        "type"             : "DIRECTORY"
      },
      ...
    ]
  }
}
```

## 8.9.2 Prerequisites

### Secure Cluster

Before accessing WebHDFS in secure mode, you need to secure the underlying Hadoop cluster, as described earlier in this chapter, starting with [Configuring Kerberos for HDFS and YARN \(MapReduce\)](#).

Note that as part of the procedure to secure your cluster, you will [edit the `site.xml` file](#). The `dfs.webhdfs.enabled` and `dfs.web.authentication.kerberos.principal` properties in this file that must be set correctly to enable secure WebHDFS.

After security is enabled, all WebHDFS operations will fail with a 401 error until you use WebHDFS in secure mode.

## 8.9.3 Configuring Secure WebHDFS

Once the cluster is secured, perform the following steps:

### Create a Principal

To access WebHDFS in secure mode, a new Kerberos user (or Principal) must be created in Kerberos. To do this, use the `kadmin.local` command on the host where Kerberos is installed, and then run the `addprinc <username>` command. For example:

```
# kadmin.local
Authenticating as principal root/admin@TESTREALM.COM with password.
kadmin.local: addprinc testuser
WARNING: no policy specified for testuser@TESTREALM.COM; defaulting to no policy
Enter password for principal "testuser@TESTREALM.COM":
Re-enter password for principal "testuser@TESTREALM.COM":
Principal "testuser@TESTREALM.COM" created.
```

### Add to Groups

#### Optional

Group information is accessed on the Namenode. If you need the Principal you just created (`testuser`, in our example above) to reside in specific groups (for example if you need permission to run a `GETCONTENTSUMMARY` command), you need to create an OS user on the Namenode that belongs to the groups you need, for example, `hadoop`.

To add a regular user on the NameNode, run the `useradd` command, as follows:

```
adduser -N -g hadoop testuser
```

## 8.9.4 Using WebHDFS in Secure Mode

To verify WebHDFS works in secure mode, perform the following steps:

### Authenticate

You must authenticate yourself as a valid Kerberos user. You do this by running `kinit` command with your user name, here `testuser`:

```
$ kinit testuser
Password for testuser@TESTREALM.COM:
```

## Verify your Authentication

### Optional

If `kinit` is successful, you will be able to validate that you have a valid Kerberos ticket using the `klist` command, as follows:

```
$ klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: testuser@TESTREALM.COM

Valid starting    Expires          Service principal
09/19/13 01:36:40 09/20/13 01:36:40  krbtgt/TESTREALM.COM@TESTREALM.COM
        renew until 09/26/13 01:36:40
```

## Verify Curl supports Kerberos Negotiate

Your version of curl must support Keberos's GSS-Negotiate feature; you can verify this by running the following:

```
$ curl -V
curl 7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8x zlib/1.2.5
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtsp smtp smtps
telnet tftp
Features: AsynchDNS GSS-Negotiate IPv6 Largefile NTLM NTLM_WB SSL libz
```

## Run Secure WebHDFS

You can now run a secure WebHDFS command.

Note the `--negotiate` parameter in curl, which turns on Kerberos negotiate.

```
curl -i --negotiate -u testuser "http://<HOST>:50070/webhdfs/v1/?op=GETCONTENTSUMMARY"
```

You should see a response like this:

```
Enter host password for user 'testuser':
HTTP/1.1 401
Date: Thu, 19 Sep 2013 01:45:55 GMT
Pragma: no-cache
Date: Thu, 19 Sep 2013 01:45:55 GMT
```

```

Pragma: no-cache
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=;Path=/;Expires=Thu, 01-Jan-1970 00:00:00 GMT
Content-Type: text/html;charset=ISO-8859-1
Cache-Control: must-revalidate,no-cache,no-store
Content-Length: 1358
Server: Jetty(7.6.10.v20130312)

HTTP/1.1 200 OK
Date: Thu, 19 Sep 2013 01:45:55 GMT
Pragma: no-cache
Cache-Control: no-cache
Date: Thu, 19 Sep 2013 01:45:55 GMT
Pragma: no-cache
Set-Cookie:
hadoop.auth="u=testuser&p=testuser@SMUNGEEREALM.COM&t=kerberos&e=1379591155709&s=z1zr9/EuqluQ9C2F6
Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(7.6.10.v20130312)

{"ContentSummary":{"directoryCount":29,"fileCount":9,"length":3156,"quota":2147483647,"spaceConsum

```

This response verifies that you are accessing WebHDFS in secure mode. Note the initial 401 response above, followed by the 200 response. This is a result of the curl Kerberos negotiation.

## 8.10 Troubleshooting

### Log Files

A good first step is to look for exceptions that may give you a hint in the logs (where "hostname" is the host the log is on):

- **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:

```
HADOOP_OPTS="$HADOOP_OPTS -Dsun.security.krb5.debug=true"
```

**Data node will not start:**

- if you are getting a message about data node 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 run: `klist -e -k -t pathtokeytab`
- Check that you modified `hadoop-env.sh` and `/etc/init.d/hadoop-hdfs-datanode` properly
- If these are correct, run `/etc/init.d/hadoop-hdfs-datanode start` and look at the output
- If there are no printed errors or it complains that no VM can be found it is a JSVC problem, see [Installing JSVC](#)

**Cannot find principal:**

- Check keytab and principal entries in site xml, keytab dir perms
- Cannot get password for username: check keytab and principal entries in site xml, keytab dir perms if these all look OK then do a `kinit -k -t /etc/security/keytab/service.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`.

## 9 Manually Upgrading PHD from 1.0.1 to 1.1 - RPM

---

### 9.1 Overview

---

PHD support only stack level upgrade, don't support component level upgrade. All PHD component should come from same release package and all PHD components need to upgrade to same release version.

For each component upgrade, the following step need to be followed:

- Stop Service
- Backup Configuration
- RPM upgrade
- Restore Configuration
- Start Service

### 9.2 Components to be Upgraded (ICM support)

---

- Zookeeper
- Bigtop utilities
- HDFS
- Yarn
- HBase
- Hive
- Pig
- Mahout

The above component upgrade is supported by ICM. Manual RPM upgrade is risky and upgrade by ICM is strongly recommended.

#### 9.2.1 Upgrade Bigtop utilities

RPM Upgrade Bigtop Utilities

#### 9.2.2 Upgrade Zookeeper

1. Stop the zookeeper-server service.
2. RPM Upgrade Zookeeper
3. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
4. Restart the zookeeper-server service on Zookeeper server.

### 9.2.3 Upgrade Hadoop

1. Stop Hadoop services.
2. RPM upgrade hadoop.
3. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
4. Restart Hadoop services

### 9.2.4 Upgrade HBase

1. Stop HBase master/region/thrift/restful service.
2. RPM upgrade Hbase.
3. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
4. Restart HBase master/region/thrift/restful service.

### 9.2.5 Upgrade Hive

1. Stop Hive/metastore services.
2. RPM upgrade Hive
3. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
4. Restart Hive/metastore services.

### 9.2.6 Upgrade Pig

1. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
2. Restore any customized UDF jars to lib directory.

### 9.2.7 Upgrade Mahout

1. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.

## 9.3 Components to be Upgraded (ICM don't support)

---

- Sqoop
- Flume

These instructions assume that the following components were already upgraded to PHD stack 1.1.0.0 and worked correctly:

- Hadoop
- Hbase

RPM upgrade requires `root` privileges on the nodes to be upgraded. Unless specified, all the following operations should be run by user `root`.

## 9.4 Upgrade Instructions

### 9.4.1 Download and extract the new PHD 1.1 tarball

1. Download the 1.1.0.0 version of PHD release tarball package, the file name is:  
`PHD-1.1.0.0-<build number>.tar.gz`.
2. Extract it to some place, the extracted file structure looks like the following:

```
$> tar xzf PHD-1.1.0.0-<build number>.tar.gz
$> cd PHD-1.1.0.0-<build number>
$> ls
README  hadoop    hcatalog  mahout    pig        utility
flume   hbase     hive      oozie     sqoop      zookeeper
```

You need to do this on every node in your cluster.

### 9.4.2 Backup old configurations

Backup your existing configuration files before you upgrade.

The easiest way to backup the configuration is to copy the whole `/etc/gphd/` folder on each node. However, to avoid invalid symbol links after backing up, we suggest copy the configuration files component by component.

We also require you back up the external jar files used to connect with database for Sqoop:

```
$> cp -r /etc/gphd/flume/conf <where to save Flume configuration>
$> cp -r /etc/gphd/sqoop/conf <where to save Sqoop configuration>
$> cp -r /usr/lib/gphd/sqoop/lib <where to save external jar files>
```

### 9.4.3 Upgrade Flume

#### 1. Stop Flume Agent Service

Before you begin the upgrade, stop the `flume-agent` service:



```
$> sudo service flume-agent stop
stopping flume-agent
```

```
[ OK ]
```

## 2. Upgrade the Flume Client

```
$> cd PHD-1.1.0.0-<build number>/flume/rpm
$> rpm -U flume-1.3.1_gphd_2_1_0_0-69.noarch.rpm
<output omitted here>
$> rpm -qa | grep flume
flume-1.3.1_gphd_2_1_0_0-69.noarch
```

## 3. Upgrade the Flume Agent

```
$> cd PHD-1.1.0.0-<build number>/flume/rpm
$> rpm -U flume-1.3.1_gphd_2_1_0_0-69.noarch.rpm
flume-agent-1.3.1_gphd_2_1_0_0-69.noarch.rpm
<output omitted here>
$> rpm -qa | grep flume
flume-1.3.1_gphd_2_1_0_0-69.noarch
flume-agent-1.3.1_gphd_2_1_0_0-69.noarch
```

## 4. Restore the old configurations

After upgrading all Flume packages, you need to restore your configuration files in `/etc/gphd/flume/conf`, especially the `flume.conf`.

Check your previous backup configuration files, and manually add any change you made back to your upgraded configuration files.

## 5. Restart Flume Agent Service

```
$> sudo service flume-agent start
starting flume-agent, logging to /var/log/gphd/flume/flume-agent.log
[ OK ]
```

# 9.4.4 Upgrade Sqoop

## 1. Stop Sqoop Metastore Service

Before you run the upgrade, first stop the `sqoop-metastore` service:

```
$> sudo service sqoop-metastore stop
stopping sqoop-metastore
```

```
[ OK ]
```

## 2. Upgrade Sqoop Client

```
$> cd PHD-1.1.0.0-<build number>/sqoop/rpm
$> rpm -U sqoop-1.4.2_gphd_2_1_0_0-69.noarch.rpm
<output omitted here>
$> rpm -qa | grep sqoop
sqoop-1.4.2_gphd_2_1_0_0-69.noarch
```

### 3. Upgrade Sqoop Metastore

```
$> cd PHD-1.1.0.0-<build number>/sqoop/rpm
$> rpm -U sqoop-1.4.2_gphd_2_1_0_0-69.noarch.rpm
sqoop-metastore-1.4.2_gphd_2_1_0_0-69.noarch.rpm
<output omitted here>
$> rpm -qa | grep sqoop
sqoop-1.4.2_gphd_2_1_0_0-69.noarch
sqoop-metastore-1.4.2_gphd_2_1_0_0-69.noarch.rpm
```

### 4. Restore old configurations

After upgrading all Sqoop packages, you need to restore your configuration files in `/etc/gphd/sqoop/conf`, especially the `sqoop-site.xml`.

Check your previous backup configuration files, and manually add any change you made back to your upgraded configuration files.

You also need to restore your external jar files to `/usr/lib/gphd/sqoop/lib`.

### 5. Restart Sqoop Metastore service

```
$> sudo service sqoop-metastore start
starting sqoop-metastore, logging to
/var/log/gphd/sqoop/sqoop-metastore-sqoop-centos62-5.log
```

[ OK ]

# 10 Manually Upgrading PHDMR1 from 1.0.1 to 1.1 - RPM

---

## 10.1 Overview

---

PHD support only stack level upgrade, don't support component level upgrade. All PHD component should come from same release package and all PHD components need to upgrade to same release version.

For each component upgrade, the following step need to be followed:

- Stop Service
- Backup Configuration
- RPM upgrade
- Restore Configuration
- Start Service

## 10.2 Components to be upgraded

---

- Zookeeper
- Bigtop utilities
- HDFS
- Hadoop MR1
- HBase
- Hive
- Pig
- Flume
- Sqoop
- Mahout

## 10.3 Prerequisites

---

These instructions assume that the you already have set up a cluster with PHDMR1 1.0.1 stack installed, and working correctly. This cluster should contain nodes (machines) installed different packages to perform different operations.

The upgrade needs to be performed on each node, the steps are similar for different kind of nodes, with slight differences according to the different packages installed.

Keep a list of the nodes in the cluster and the packages installed on each node, perform correct upgrading operations according to the node function.

RPM upgrade requires `root` privileges on the nodes to be upgraded. Unless specified, all the following operations should be run by user `root`.

## 10.4 Upgrade Instructions



The file names, commands and outputs listed in this section are for your reference only. Due to environment differences, they will be similar to, but not exactly the same as what you will get when you perform the upgrade process in your own environment.

### 10.4.1 Download and extract the new PHDMR1 1.1 tarball

1. Download the 1.1 version of PHDMR1 release tarball package, the file name should be: `PHDMR1-1.1.0.0.tar.gz`.
2. Extract it to some place, the extracted file structure looks like the following:

```
$> tar xzf PHDMR1-1.1.0.0.tar.gz
$> cd PHDMR1-1.1.0.0
$> ls
flume      hadoop-mr1  hcatalog   mahout     README    utility
hadoop     hbase      hive       pig        sqoop     zookeeper
```

You need to do this on every node in your cluster.

### 10.4.2 Backup old configurations

Backup your existing configuration files before you upgrade.

The easiest way to backup the configuration is to copy the whole `/etc/gphd/` folder on each node. However, to avoid invalid symbol links after backing up, we suggest copy the configuration files component by component.

We recommend you create folders for each component's backup configuration, in the following code snippet, we assume you put the configurations in folder `~/phdmr1_backup/`, and create sub-folders for each component under it. Adjust the target path according to your actual environment.

```
$> cp -r /etc/gphd/hadoop/conf/* ~/phdmr1_backup/hadoop/conf/
$> cp -r /etc/gphd/zookeeper/conf/* ~/phdmr1_backup/zookeeper/conf/
$> cp -r /etc/gphd/hbase/conf/* ~/phdmr1_backup/hbase/conf/
$> cp -r /etc/gphd/hive/conf/* ~/phdmr1_backup/hive/conf/
$> cp -r /etc/gphd/pig/conf/* ~/phdmr1_backup/pig/conf/
$> cp -r /etc/gphd/flume/conf/* ~/phdmr1_backup/flume/conf/
$> cp -r /etc/gphd/sqoop/conf/* ~/phdmr1_backup/sqoop/conf/
$> cp -r /etc/gphd/mahout/conf/* ~/phdmr1_backup/mahout/conf/
```

## 10.4.3 Upgrade Zookeeper

### 1. Stop service

Before you begin the upgrade, stop the `zookeeper-server` service.

```
$> service zookeeper-server stop
stopping , logging to /var/log/gphd/zookeeper/zookeeper.out
JMX enabled by default
Using config: /etc/gphd/zookeeper/conf/zoo.cfg
Stopping zookeeper ... STOPPED
```

[ OK ]

### 2. Upgrade Zookeeper

#### 1. On Zookeeper Server:

Go to the directory where you extracted PHDMR1 1.1, navigate to the Zookeeper RPMs directory.

```
$> cd PHDMR1-1.1.0.0/zookeeper/rpm
$> rpm -U zookeeper-3.4.5_gphd_2_1_0_0-67.noarch.rpm
zookeeper-server-3.4.5_gphd_2_1_0_0-67.noarch.rpm
../../utility/rpm/bigtop-utils-0.4.0_gphd_2_1_0_0-69.noarch.rpm
```

Repeat the above steps on each node installed Zookeeper server.

#### 2. On Zookeeper Client:

If you have only Zookeeper client installed (on non-Zookeeper server nodes), only one package need to be upgraded.

```
$> cd PHDMR1-1.1.0.0/zookeeper/rpm
$> rpm -U zookeeper-3.4.5_gphd_2_1_0_0-67.noarch.rpm
```

Repeat the above steps on each node installed Zookeeper client.

### 3. Restore old configurations

After upgrading Zookeeper, you need to restore your configuration files in `/etc/gphd/zookeeper/conf`, especially the file `zoo.cfg`, on all Zookeeper server and clients nodes. Check your previous backup configuration files, and manually add any change you made back to your upgraded configuration files.

### 4. Restart Zookeeper service

Restart the `zookeeper-server` service on Zookeeper server.

```
$> service zookeeper-server start
starting zookeeper-server, logging to /var/log/gphd/zookeeper/zookeeper.out
JMX enabled by default
```

```
Using config: /etc/gphd/zookeeper/conf/zoo.cfg
Starting zookeeper ... STARTED
```

```
[ OK ]
```

## 10.4.4 Upgrade Bigtop Utilities

Bigtop utilities are needed by many other packages, therefore it's good practice to upgrade them before you upgrade HDFS and Hadoop MR1 components.

### 1. Identify packages to update

Not every node in your cluster has all Bigtop utility packages installed. You should only upgrade those packages you previously had installed. To identify which packages are installed:

On each node of your cluster, run:

```
$> rpm -qa | grep bigtop
bigtop-jsvc-1.0.15_gphd_2_0_1_0-43.x86_64
bigtop-utils-0.4.0_gphd_2_0_1_0-43.noarch
```

Each line in the output of this command means a Bigtop utility package you need to upgrade, if no output is returned, you don't need to upgrade Bigtop utilities on this node.

### 2. Upgrade Bigtop

On a node where Bigtop utilities were installed, run:

```
$> cd PHDMM1-1.1.0.0/utility/rpm
$> rpm -U bigtop-jsvc-1.0.15_gphd_2_1_0_0-69.x86_64.rpm
bigtop-utils-0.4.0_gphd_2_1_0_0-69.noarch.rpm
```

You need to ensure the names of the rpm packages listed in the above command match the packages you identified in the last step, that is, if you only have `bigtop-jsvc` package installed, you should only list `bigtop-jsvc-1.0.15_gphd_2_1_0_0-69.x86_64.rpm` in the above command, if you only have `bigtop-utils` package installed, you should only list `bigtop-utils-0.4.0_gphd_2_1_0_0-69.noarch.rpm` in the above command, if you have both installed, list both.

Repeat the above steps on every node you have Bigtop utilities installed.

## 10.4.5 Upgrade HDFS

Upgrade HDFS means upgrade HDFS packages on name node, secondary name node (if any) and each data node.

### 1. Stop Services

Before you begin the upgrade, stop the corresponding HDFS services on each node.

1. On the NameNode

```
$> service hadoop-hdfs-namenode stop
stopping namenode
```

```
[ OK ]
```

## 2. On Secondary NameNode

```
$> service hadoop-hdfs-secondarynamenode stop
stopping secondarynamenode
```

```
[ OK ]
```

## 3. On each Data Node

```
$> service hadoop-hdfs-datanode stop
stopping datanode
```

```
[ OK ]
```

## 2. Upgrade HDFS

### 1. On the NameNode:

Because HDFS depends on hadoop package, which depends on bigtop utilities; you should upgrade hadoop packages, bigtop utility packages together with HDFS packages.

```
$> cd PHDMR1-1.1.0.0/hadoop/rpm
$> rpm -U hadoop-2.0.5_alpha_gphd_2_1_0_0-68.x86_64.rpm
hadoop-hdfs-2.0.5_alpha_gphd_2_1_0_0-68.x86_64.rpm
hadoop-hdfs-namenode-2.0.5_alpha_gphd_2_1_0_0-68.x86_64.rpm
```

### 2. On Secondary NameNode:

Similar to NameNode, you need to upgrade hadoop package, bigtop utility packages and HDFS secondary name node package.

```
$> cd PHDMR1-1.1.0.0/hadoop/rpm
$> rpm -U hadoop-2.0.5_alpha_gphd_2_1_0_0-68.x86_64.rpm
hadoop-hdfs-2.0.5_alpha_gphd_2_1_0_0-68.x86_64.rpm
hadoop-hdfs-secondarynamenode-2.0.5_alpha_gphd_2_1_0_0-68.x86_64.rpm
```

### 3. On DataNode:

Similar to name node, you need to upgrade hadoop package, bigtop utility packages and HDFS data node package

```
$> cd PHDMR1-1.1.0.0/hadoop/rpm
$> rpm -U hadoop-2.0.5_alpha_gphd_2_1_0_0-68.x86_64.rpm
hadoop-hdfs-2.0.5_alpha_gphd_2_1_0_0-68.x86_64.rpm
hadoop-hdfs-datanode-2.0.5_alpha_gphd_2_1_0_0-68.x86_64.rpm
```

## 3. Restore old configurations

After upgrading all HDFS packages, you will need to restore your configuration files in `/etc/gphd/hadoop/conf`, especially the `core-site.xml` and `hdfs-site.xml`.

Check your previous backup configuration files, and manually add any change you made back to your upgraded configuration files.

You need to do this on NameNode, secondary NameNode and each DataNode.



Due to configuration files containing many user customized information and being version dependent, the backup and restore process cannot be completed automated so far, thus manually work is required to update these files.

#### 4. Restart services



Please ensure you have already upgraded all HDFS nodes (name node, secondary name node, all data nodes) using the same version of PHDMR1 package and restored correct configurations on all of them before restarting service. Otherwise, HDFS may not function properly.

##### 1. On NameNode

```
$> service hadoop-hdfs-namenode start
starting namenode, logging to
/var/log/gphd/hadoop-hdfs/hadoop-hdfs-namenode-hdsh175.out
[ OK ]
```

##### 2. On Secondary NameNode:

```
$> service hadoop-hdfs-secondarynamenode start
starting secondarynamenode, logging to
/var/log/gphd/hadoop-hdfs/hadoop-hdfs-secondarynamenode-hdsh176.out
[ OK ]
```

##### 3. On each DataNode:

```
$> service hadoop-hdfs-datanode start
starting datanode, logging to
/var/log/gphd/hadoop-hdfs/hadoop-hdfs-datanode-hdsh177.out
[ OK ]
```

## 10.4.6 Upgrade Hadoop MR1



Upgrade Hadoop MR1 means upgrade Hadoop MR1 job tracker node and each Hadoop MR1 task tracker node.

### 1. Stop services

1. On the job tracker node:

```
$> service hadoop-mr1-jobtracker stop
stopping jobtracker
[ OK ]
```

2. On each task tracker node:

```
$> service hadoop-mr1-tasktracker stop
stopping tasktracker
[ OK ]
```

### 2. Upgrade Hadoop MR1

1. On the job tracker node:

```
$> cd PHDMR1-1.1.0.0/hadoop-mr1/rpm
$> rpm -U hadoop-mr1-1.0.3_gphd_2_1_0_0-11.x86_64.rpm
hadoop-mr1-jobtracker-1.0.3_gphd_2_1_0_0-11.x86_64.rpm
```

2. On the task tracker node:

```
$> cd PHDMR1-1.1.0.0/hadoop-mr1/rpm
$> rpm -U hadoop-mr1-1.0.3_gphd_2_1_0_0-11.x86_64.rpm
hadoop-mr1-tasktracker-1.0.3_gphd_2_1_0_0-11.x86_64.rpm
```

### 3. Restore old configurations

After upgrading, you need to restore your configuration files in `/etc/gphd/hadoop/conf`, especially the file `mapred-site.xml` on the job tracker node and each task tracker node. Check your previous backup configuration files, and manually add any change you made back to your upgraded configuration files.

### 4. Restart Services

1. On the job tracker node:

```
$> service hadoop-mr1-jobtracker start
starting jobtracker, logging to
/var/log/gphd/hadoop-mr1/hadoop-mapred-jobtracker-hdsh176.out
[ OK ]
```

2. On task tracker node:

```
$> service hadoop-mr1-tasktracker start
starting tasktracker, logging to
/var/log/gphd/hadoop-mr1/hadoop-mapred-tasktracker-hdsh177.out
[ OK ]
```

## 10.4.7 Upgrade HBase

### 1. Stop services

#### 1. On HBase master server:

```
$> service hbase-master stop
stopping master.
[ OK ]
```

#### 2. On HBase region server:

```
$> service hbase-regionserver stop
stopping regionserver.
[ OK ]
```

#### 3. On HBase Thrift server:

```
$> service hbase-thrift stop
stopping thrift.
[ OK ]
```

#### 4. On HBase Restful server:

```
$> service hbase-rest stop
stopping rest.
[ OK ]
```

### 2. Upgrade HBase

#### 1. On HBase master server:

```
$> cd PHDMR1-1.1.0.0/hbase/rpm
$> rpm -U hbase-0.94.8_gphd_2_1_0_0-11.noarch.rpm
hbase-master-0.94.8_gphd_2_1_0_0-11.noarch.rpm
```

#### 2. On HBase region server:

```
$> cd PHDMR1-1.1.0.0/hbase/rpm
$> rpm -U hbase-0.94.8_gphd_2_1_0_0-11.noarch.rpm
hbase-regionserver-0.94.8_gphd_2_1_0_0-11.noarch.rpm
```

### 3. On HBase Thrift server:

```
$> cd PHDMR1-1.1.0.0/hbase/rpm
$> rpm -U hbase-thrift-0.94.8_gphd_2_1_0_0-11.noarch.rpm
hbase-0.94.8_gphd_2_1_0_0-11.noarch.rpm
```

### 4. On HBase Restful server:

```
$> cd PHDMR1-1.1.0.0/hbase/rpm
$> rpm -U hbase-0.94.8_gphd_2_1_0_0-11.noarch.rpm
hbase-rest-0.94.8_gphd_2_1_0_0-11.noarch.rpm
```

## 3. Restore old configurations

After upgrading HBase packages on all your HBase nodes (all master servers, region servers, Thrift server and Restful servers), on each node, check the configuration files in `/etc/gphd/hbase/conf`, especially the file `hbase-site.xml`, contains the correct configuration change you made before upgrading. If not correct, manually fix them from your previous backup copy of configuration files.

## 4. Restart services

After upgrading all HBase nodes, and restore configurations on each node, restart all the services:

### 1. On HBase master server:

```
$> service hbase-master start
starting master, logging to /var/log/gphd/hbase/hbase-hbase-master-hdsh175.out
[ OK ]
```

### 2. On HBase region server:

```
$> service hbase-regionserver start
starting regionserver, logging to
/var/log/gphd/hbase/hbase-hbase-regionserver-hdsh177.out
[ OK ]
```

### 3. On HBase Thrift server:

```
$> service hbase-thrift start
starting thrift, logging to /var/log/gphd/hbase/hbase-hbase-thrift-hdsh176.out
[ OK ]
```

### 4. On HBase Restful server:

```
$> service hbase-rest start
starting rest, logging to /var/log/gphd/hbase/hbase-hbase-rest-hdsh176.out
[ OK ]
```

## 10.4.8 Upgrade Hive

### 1. Stop services

#### 1. On Hive server:

```
$> service hive-server stop
stopping hive-server
[ OK ]
```

#### 2. On Hive metastore server:

```
$> service hive-metastore stop
stopping hive-metastore
[ OK ]
```

### 2. Upgrade Hive

#### 1. On Hive server:

```
$> cd PHDMR1-1.1.0.0/hive/rpm
$> rpm -U hive-0.11.0_gphd_2_1_0-11.noarch.rpm
hive-server-0.11.0_gphd_2_1_0-11.noarch.rpm
```



In PHDMR1 1.1 version, there is a new package called `hive-server2`, which intends to replace old `hive-server` package. This fact provides you another choice besides upgrading `hive-server`, that is, you can uninstall `hive-server` then install `hive-server2`. However, running `hive-server` and `hive-server2` at same time on the same machine may cause errors and malfunctions. Though by carefully configuration it's possible to make them work together on the same machine, we recommend avoid doing so.

#### 2. On Hive metastore server:

```
$> cd PHDMR1-1.1.0.0/hive/rpm
$> rpm -U hive-0.11.0_gphd_2_1_0-11.noarch.rpm
hive-metastore-0.11.0_gphd_2_1_0-11.noarch.rpm
```

### 3. Restore old configurations

After you upgraded all your Hive nodes, on each node, check the configuration files in `/etc/gphd/hive/conf`, especially the file `hive-site.xml`, contains the correct configuration change you made before upgrading. If not correct, manually fix them from your previous backup copy of configuration files.

### 4. Restart services

#### 1. On Hive server:

```
$> service hive-server start
starting hive-server, logging to /var/log/gphd/hive/hive-server.log
[ OK ]
```

2. On Hive metastore server:

```
$> service hive-metastore start
starting hive-metastore, logging to /var/log/gphd/hive/hive-metastore.log
[ OK ]
```

## 10.4.9 Upgrade Pig

### 1. Upgrade Pig

On a machine installed pig:

```
$> cd PHDMR1-1.1.0.0/pig/rpm
$> rpm -U pig-0.10.1_gphd_2_1_0_0-11.noarch.rpm
```

### 2. Restore old configurations

After you upgraded pig, check the configuration files in `/etc/gphd/pig/conf`, especially the file `pig.properties`, contains the correct configuration change you made before upgrading. If not correct, manually fix them from your previous backup copy of configuration files.

## 10.4.10 Upgrade Flume

### 1. Stop Flume Agent Service [Optional]

Before you upgrade, first stop the `flume-agent` service:

```
$> sudo service flume-agent stop
stopping flume-agent
[ OK ]
```

### 2. Upgrade Flume client

```
$> cd PHDMR1-1.1.0.0/flume/rpm
$> rpm -U flume-1.3.1_gphd_2_1_0_0-69.noarch.rpm
```

### 3. Upgrade Flume Agent [Optional]

```
$> cd PHDMR1-1.1.0.0/flume/rpm
$> rpm -U flume-1.3.1_gphd_2_1_0_0-69.noarch.rpm
flume-agent-1.3.1_gphd_2_1_0_0-69.noarch.rpm
```

#### 4. Restore old configurations

After upgrading all Flume packages, you will need to restore your configuration files in `/etc/gphd/flume/conf`, especially the `flume.conf`.

Check your previous backup configuration files, and manually add any change you made back to your upgraded configuration files.

#### 5. Restart Flume agent service [Optional]

```
$> sudo service flume-agent start
starting flume-agent, logging to /var/log/gphd/flume/flume-agent.log
[ OK ]
```

## 10.4.11 Upgrade Sqoop

#### 1. Stop Sqoop Metastore Service [Optional]

Before you upgrade, first stop the `sqoop-metastore` service:

```
$> sudo service sqoop-metastore stop
stopping sqoop-metastore
[ OK ]
```

#### 2. Upgrade Sqoop client

```
$> cd PHDMR1-1.1.0.0/sqoop/rpm
$> rpm -U sqoop-1.4.2_gphd_2_1_0_0-69.noarch.rpm
```

#### 3. Upgrade Sqoop metastore [Optional]

```
$> cd PHDMR1-1.1.0.0/sqoop/rpm
$> rpm -U sqoop-1.4.2_gphd_2_1_0_0-69.noarch.rpm
sqoop-metastore-1.4.2_gphd_2_1_0_0-69.noarch.rpm
```

#### 4. Restore old configurations

After upgrading all Sqoop packages, you will need to restore your configuration files in `/etc/gphd/sqoop/conf`, especially the `sqoop-site.xml`.

Check your previous backup configuration files, and manually add any change you made back to your upgraded configuration files.

#### 5. Restart Sqoop metastore service [Optional]

```
$> sudo service sqoop-metastore start
starting sqoop-metastore, logging to
/var/log/gphd/sqoop/sqoop-metastore-sqoop-centos62-5.log
[ OK ]
```

## 10.4.12 Upgrade Mahout

### 1. Upgrade Mahout

On the machine installed mahout:

```
$> cd PHDMR1-1.1.0.0/mahout/rpm
$> rpm -U mahout-0.7_gphd_2_1_0_0-11.noarch.rpm
```

### 2. Restore old configurations

After you upgraded Mahout packages, check the configuration files in `/etc/gphd/mahout/conf`, check each `.props` file contains the correct configuration change you made before upgrading. If not correct, manually fix them from your previous backup copy of configuration files.

# 11 Manually Upgrading PHD/PHDMR1 from 1.0.1 to 1.1 - Binary

---

## 11.1 Overview

---

PHD supports stack level upgrade and does not support component level upgrade. All PHD component should come from same package and all PHD components need to upgrade to same version.

Note: This section includes upgrade instructions for both Yarn-based clusters and MR1-based clusters.

For each component upgrade, the following steps need to follow:

- Stop Service
- Backup Configuration
- Binary Upgrade
- Restore Configuration
- Restart Service

## 11.2 Components to be Upgraded

---

- Zookeeper
- HDFS
- Hadoop Yarn
- Hadoop MR1
- HBase
- Hive
- Pig
- Flume
- Sqoop
- Mahout

## 11.3 Prerequisites

---

You should have already set up a cluster-installed PHD/PHDMR1 stack version 1.0.1, and it should be working properly.

## 11.4 Upgrade Zookeeper

---

You can perform a rolling upgrade for a Zookeeper cluster. For each node, follow the steps below to upgrade zookeeper service. Make sure the new upgraded node has joined the quorum, before starting to upgrade the next node.



1. Stop the `zookeeper-server` service.
2. Unpack the Zookeeper tarball `zookeeper-3.4.5-gphd-2.1.0.0.tar.gz`
3. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
4. Update environment variables including `ZK_HOME` according to new directory and make sure environment variables take effect.
5. Restart the `zookeeper-server` service on Zookeeper server.

## 11.5 Upgrade Hadoop

---

1. Stop Yarn (for Yarn) or Mapreduce (for MR1) services.
2. Stop HDFS services.
3. Unpack Hadoop tar file `hadoop-2.0.5-alpha-gphd-2.1.0.0.tar.gz` to new directory on each node.
4. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
5. Update environment variables including `HADOOP_HOME`, `HADOOP_COMMON_HOME`, `HADOOP_HDFS_HOME`, `HADOOP_MAPRED_HOME` according to new directory and make sure environment variables take effect.
6. Restart HDFS services.
7. Restart Yarn (Yarn) or Mapreduce (MR1) services.

## 11.6 Upgrade HBase

---

1. Stop HBase master/region/thrift/restful service.
2. Unpack HBase tar file `hbase-0.94.8-gphd-2.1.1.0.tar.gz` to new directory.
3. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
4. Update environment variables including `HBASE_HOME` according to new directory and make sure environment variables take effect.
5. Restart HBase master/region/thrift/restful service.

## 11.7 Upgrade Hive

---

1. Stop Hive/metastore services.
2. Unpack the Hive tarball `hive-0.11.0-gphd-2.1.1.0.tar.gz` to new directory.
3. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
4. Update environment variables including `HIVE_HOME` according to new directory and make sure environment variables take effect.
5. Restart Hive/metastore services.

## 11.8 Upgrade Pig

---

1. Unpack the Hive tarball `pig-0.10.1-gphd-2.1.1.0.tar.gz`
2. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
3. Update environment variables including `PIG_HOME` according to new directory and make sure environment variables take effect.
4. Restore any customized UDF jars to lib directory.

## 11.9 Upgrade Flume

---

1. Stop Flume agent service.
2. Unpack the Flume tarball `flume-1.3.1-gphd-2.1.1.0-bin.tar.gz`
3. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
4. Update environment variables including `FLUME_HOME` according to new directory and make sure environment variables take effect.
5. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
6. Restart flume agent service.

## 11.10 Upgrade Sqoop

---

1. Stop the `sqoop-metastore` service.
2. Unpack the Sqoop  
`sqoop-1.4.2-gphd-2.0.3.0.bin__hadoop-2.0.5-alpha-gphd-2.0.3.0.tar.gz`
3. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
4. Update environment variables including `SQOOP_HOME` according to new directory and make sure environment variables take effect.
5. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
6. Restart the `sqoop-metastore` service

## 11.11 Upgrade Mahout

---

1. Unpack Mahout `mahout-distribution-0.7-gphd-2.1.0.0.tar.gz`
2. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.
3. Update environment variables including `MAHOUT_HOME` according to new directory and make sure environment variables take effect.

4. Restore configuration from original installed directory and make sure configuration is compatible with new installed version.