

Pivotal HD Enterprise

Version 2.0

Stack and Tools Reference

Rev: A03 – May 7, 2014

Copyright

Copyright © 2014 Pivotal Software, Inc. All Rights reserved.

Pivotal Software, 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." Pivotal Software, 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.

About Pivotal Software, Inc.

Greenplum transitioned to a new corporate identity (Pivotal, Inc.) in 2013. As a result of this transition, there will be some legacy instances of our former corporate identity (Greenplum) appearing in our products and documentation. If you have any questions or concerns, please do not hesitate to contact us through our web site: <http://gopivotal.com/about-pivotal/support>.

Contents

Chapter 1. Overview of Apache Stack and Pivotal Components	9
Deployment/Installation Options	10
Pivotal Command Center Deployment	10
Manual Installation	10
Core Apache Stack Components	11
Pivotal Other Components	12
Location of Installation Instructions	13
Hadoop Versions	14
How to Use this Document	15
 Chapter 2. Manually Installing Pivotal HD 2.0 Stack	 16
Distribution Contents	17
Accessing PHD 2.0	18
Prerequisites	18
Installation Notes	19
Hadoop HDFS	20
Hadoop HDFS RPM Packages	20
Prerequisites: Core Package Setup	23
HDFS Namenode Setup	23
HDFS Datanode Setup	23
HDFS Secondary Namenode Setup	23
HDFS NFS Gateway Setup	23
HDFS Configuration	24
Usage	24
Hadoop YARN	28
Hadoop YARN RPM Packages	28
Prerequisites: Core Package Setup	30
YARN ResourceManager Setup	30
YARN NodeManager Setup	30
Mapreduce HistoryServer Setup	30
YARN ProxyServer Setup	30
YARN Configuration	31
YARN Usage	31
Hadoop Pseudo-distributed Configuration	34
Hadoop Pseudo-distributed Configuration Setup	34
Zookeeper	35
Zookeeper RPM Packages	35
Zookeeper Server Setup	36

Zookeeper Client Setup	36
Zookeeper Configuration	36
Usage	36
HBase	38
Prerequisites	38
HBase RPM Packages	38
HBase Master Setup	39
HBase RegionServer Setup	39
HBase Client Setup	39
HBase Thrift Server Setup	39
REST Server Setup	40
HBase Configuration	40
HBase Post-Installation Configuration	40
Usage	40
Hive	44
Hive Components	44
Prerequisites	44
Hive RPM Packages	44
Installing DB for Hive Metastore	45
Hive MetaStore Server Setup	46
Hive Server Setup	47
Hive Server2 Setup	47
Hive Configuration	48
Hive Post-installation Configuration	48
Hive Client Setup	49
Hive Usage	49
Hcatalog	51
Prerequisites	51
Hcatalog RPM Packages	51
Hcatalog Client Setup	52
Hcatalog Server Setup	52
Webhcat Setup	52
Webhcat Server Setup	52
Hcatalog Configuration	52
Usage	53
Pig	55
Prerequisites	55
Pig RPM Packages	55
Pig Client Setup	55
Pig Configuration	55
Usage	56
Piggybank Usage	56
Mahout	58
Prerequisites	58
Mahout RPM Packages	58

Mahout Client Setup	58
Mahout Configuration	58
Usage	58
Flume	60
Prerequisites	60
Flume RPM Packages	60
Flume Client Setup	60
Flume Agent Setup	61
Flume Agent Configuration	61
Usage	61
Sqoop	64
Prerequisites	64
Sqoop RPM Packages	64
Sqoop Client Setup	64
Sqoop Metastore Setup	64
Sqoop Metastore Configuration	65
Usage	65
Oozie	67
Prerequisites	67
Oozie RPM Packages	67
Oozie client Setup	67
Oozie server Setup [Optional]	68
Oozie Configuration	68
Oozie environment configuration	68
Setup Database	68
Usage	69
Oozie in HA Mode - Best Practices	74
Hamster	76
Prerequisites	76
Hamster RPM Packages	76
Hamster-core Setup	76
OpenMPI Setup	77
Hamster-rte Setup	77
Running an example	77
Hamster Usage	78
GraphLab	80
Prerequisites	80
GraphLab RPM Packages	80
Running an example	81
Usage	81
YARN configurations for better using GraphLab on Hamster	82
TroubleShooting	83

Chapter 3. Manually Upgrading Pivotal HD Stack from 1.1.1 to 2.0	85
General Upgrade Tasks for Each Component	86
Accessing the PHD 2.0 RPM	87
PHD RPM Upgrade Limitations from PHD 1.1.1 to PHD 2.0	88
Components to be Upgraded (ICM supported)	89
Upgrade Bigtop Utilities	89
Upgrade Zookeeper	89
Upgrade Hadoop	89
Upgrade HBase	90
Upgrade Hive	90
Upgrade Pig	90
Upgrade Mahout	91
Components to be Upgraded (ICM not supported)	92
Upgrade Flume	92
Upgrade Sqoop	93
Upgrade Oozie	94
Upgrade HCatalog	95
 Chapter 4. Pivotal Hadoop Enhancements	 97
HDFS Off-Cluster Client Rack Awareness	99
Usage	101
Vaidya	102
Overview	102
Installing Vaidya Files	102
Enabling Vaidya	102
Disabling Vaidya	103
Using Vaidya to Analyze Jobs	103
Vaidya Configuration Rules	103
HVE Topology Awareness	108
Topology Awareness Configuration and Verification	108
HVE Elasticity	111
Overview	111
Function List	111
Configuration	111
Command Line Interface for YARN cluster	112
 Chapter 5. Security	 114
Configuring Kerberos for HDFS and YARN (MapReduce)	118
Kerberos Set-up	118
Java Support Items Installation	122
Container and Script Modifications	123
Site XML Changes	125

Complete the HDFS/YARN Secure Configuration	130
Turning Secure Mode Off	131
Building and Installing JSVC	132
Installing the MIT Kerberos 5 KDC	133
Configuring Kerberos for HDFS High Availability	136
Zookeeper Secure Configuration	137
Zookeeper Servers	137
Zookeeper Clients	138
HBase Secure Configuration	141
HBase Master and RegionServers	141
HBase Clients	142
HBase with Secure Zookeeper Configuration	144
Access Control and PXF External Tables	144
Hive Secure Configuration	148
Changing to Hive Server 2	149
Hive warehouse permissions issues	150
Connecting and using secure Hive with Beeline	150
Configure HCatalog (WebHCat) on secure Hive	152
Prerequisites	152
Create keytab file for the WebHCat server	152
Distribute the keytab file to the WebHCat server	152
Configure WebHCat and proxy users	152
Verify WebHCat is working	153
HAWQ on Secure HDFS	155
Requirements	155
Preparation	155
Configuration	155
Troubleshooting	177
Auditing	159
Secure Web Access	161
Overview	161
Prerequisites	162
Configuring Secure WebHDFS	162
Using WebHDFS in Secure Mode	163
Secure HDFS web access via HttpFS	165
Prerequisites	165
Add principal for HttpFS	165
Create and distribute keytab	165
Set the keytab file ownership and permissions	165
Configuration	166
Restart HttpFS	166
Verify it's working	167
Flume Security Configuration	168
Prerequisites	168
Create the Flume Principal	168

Create the Flume Keytab Files	168
Distribute the Flume Keytab Files to the Flume server and change the ownership and permission	168
A single user for all HDFS sinks	168
Different users across multiple HDFS sinks	169
Oozie Security Configuration	172
Prerequisites	172
Create the Oozie Principal	172
Create the HTTP Principal for the Oozie Server	172
Create the Oozie Keytab Files	172
Copy the Oozie Keytab Files to the Oozie server and change the ownership and permission	172
Edit the Oozie Configuration	173
Using Oozie with a Secure Hive Metastore Server	173
Verify Secure Oozie	174
Sqoop Security Configuration	175
Pig Security Configuration	176
Mahout Security Configuration	177
Troubleshooting	178

Chapter 1 Overview of Apache Stack and Pivotal Components

Pivotal HD Enterprise is an enterprise-capable, commercially supported distribution of Apache Hadoop packages targeted to traditional Hadoop deployments.

Topics:

- Deployment/Installation Options
 - Pivotal Command Center Deployment
 - Manual Installation
- Core Apache Stack Components
- Pivotal Other Components
- Location of Installation Instructions
- Hadoop Versions
- How to Use this Document

Deployment/Installation Options

Pivotal HD Enterprise supports two options for deployment/installation:

- Deployment through Pivotal Command Center
- Manual installation of RPMs

Pivotal Command Center Deployment

Pivotal Command Center (PCC) is a Web-based interface that supports performing the following actions for Pivotal HD Enterprise:

- Monitoring and management of a Pivotal HD (PHD) environment.
- Deployment and configuration of PHD clusters.

PCC also provides a CLI (Command-Line Interface) which can be used to deploy and configure PHD clusters. You can deploy *most* of the components through the PCC GUI or CLI. However, note that some components (specifically Flume, Sqoop, Oozie, Hamster and GraphLab) can *only* be installed manually.

- For more information about using the GUI, see *Pivotal Command Center User Guide*.
- For more information about using the CLI, see *Pivotal HD Enterprise Installation and Administrator Guide*.

Manual Installation

All the Hadoop and Pivotal components *can* be installed manually without using the PCC GUI or CLI. For manual installs, Pivotal HD Enterprise provides RPM files.

This document provides detailed manual instructions for installing all the Apache components, as well as some of the additional components.

The following sections provide more details about each component, including the various supported deployment options for each component.

Core Apache Stack Components

Component	Description	CLI/GUI Install	Manual Install (RPM)
Hadoop	HDFS: A Hadoop distributed file system (HDFS). YARN: Next-generation Hadoop data-processing framework.	✓	✓
Pig	Procedural language that abstracts lower level MapReduce.	✓	✓
Hive	Data warehouse infrastructure built on top of Hadoop.	✓	✓
HBase	Database for random real time read/write access.	✓	✓
Hcatalog	HCatalog is a table and storage management layer for Hadoop that enables users with different data processing tools — e.g. Pig, MapReduce — to more easily read and write data on the grid.	✓	✓
Mahout	Scalable machine learning and data mining library.	✓	✓
Zookeeper	Hadoop centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.	✓	✓
Flume	A tool used for collecting and aggregating data from multiple sources to a centralized data store.		✓
Sqoop	A tool for transferring bulk data between Apache Hadoop and structured datastores.		✓
Oozie	A workflow scheduler system to manage Apache Hadoop jobs. Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions. Oozie Coordinator jobs are recurrent Oozie Workflow jobs triggered by time (frequency) and data availability.		✓

Pivotal Other Components

Component	Description	CLI/GUI Install	Manual Install (RPM)
Pivotal Command Center (installs the CLI)	A command line and web-based tool for installing, managing and monitoring your Pivotal HD cluster.		✓
Pivotal ADS - HAWQ	HAWQ is a parallel SQL query engine that combines the merits of the Greenplum Database Massively Parallel Processing (MPP) relational database engine and the Hadoop parallel processing framework.	✓	✓
Pivotal ADS - PXF	Extensibility layer to provide support for external data formats such as HBase and Hive.	✓	✓
Hamster	Developed by Pivotal, Hamster is a framework which enable users running MPI programs on Apache Hadoop YARN platform. (OpenMPI is a A High Performance Message Passing Library)		✓
GraphLab	GraphLab is a powerful new system for designing and implementing parallel algorithms in machine learning. It is a graph-based, high performance, distributed computation framework written in C++ that makes use of MPI and has its own programming model.		✓

Location of Installation Instructions

You can find installation instructions for the above components in these documents:

Component	GUI Install	CLI Install	Manual Install
Pivotal Command Center	NOT SUPPORTED	<i>Pivotal HD Enterprise Installation and Administrator Guide</i>	NOT SUPPORTED
Pivotal Hadoop Stack	<i>Pivotal Command Center User Guide</i>	<i>Pivotal HD Enterprise Installation and Administrator Guide</i>	This guide.
Pivotal ADS (HAWQ and PXF)	<i>Pivotal Command Center User Guide</i>	<i>Pivotal HD Enterprise Installation and Administrator Guide</i>	<i>Pivotal Extension Framework Installation and User Guide</i>

Hadoop Versions

PHD 2.0 is based upon Hadoop 2.2.0.

How to Use this Document

Manual Installation Instructions:

- Manually installing a YARN-based cluster using the RPM distribution? See [Manually Installing Pivotal HD 2.0 Stack](#).

Upgrade Instructions:

- Manually upgrading a YARN-based cluster using the RPM distribution? See [Manually Upgrading Pivotal HD Stack from 1.1.1 to 2.0](#).

Security:

- Securing your cluster via Kerberos? See [Security](#).

Chapter 2 Manually Installing Pivotal HD 2.0 Stack

This section describes how to manually install all the components included with Pivotal HD 2.0.

Topics:

- Distribution Contents
- Accessing PHD 2.0
- Hadoop HDFS
- Hadoop YARN
- Hadoop Pseudo-distributed Configuration
- Zookeeper
- HBase
- Hive
- Hcatalog
- Pig
- Mahout
- Flume
- Sqoop
- Oozie
- Hamster
- GraphLab

Distribution Contents

Pivotal HD 2.0 is a full Apache Hadoop distribution with Pivotal add-ons and a native integration with the Pivotal Greenplum database.

The RPM distribution of PHD 2.0 contains the following:

- **Hadoop 2.2.0**
- **Pig 0.12.0**
- **Zookeeper 3.4.5**
- **HBase 0.96.0**
- **Hive 0.12.0**
- **Hcatalog 0.12.0**
- **Mahout 0.7**
- **Flume 1.4.0**
- **Sqoop 1.4.2**
- **Oozie 4.0.0**
- **Hamster 1.0**
- **GraphLab 2.2**

Accessing PHD 2.0

Download and extract the PHD package to your working directory:

```
$> tar zxvf PHD-2.0.1.0-<nn>.tar.gz
$> ls -p PHD-2.0.1.0-<nn>
flume/      hadoop/    hbase/     hive/      oozie/          pig/    utility/
graphlab/   hamster/   hcatalog/  mahout/    open_source_licenses_PHD.txt sqoop/  zookeeper/
```

We define the replaced string which we use in the following sections for each component as follows:

Component	PHD Version	Replaced String
Hadoop	2.2.0_gphd_3_0_1_0	<PHD_HADOOP_VERSION>
HBase	0.96.0_gphd_3_0_1_0	<PHD_HBASE_VERSION>
Hive	0.12.0_gphd_3_0_1_0	<PHD_HIVE_VERSION>
Pig	0.12.0_gphd_3_0_1_0	<PHD_PIG_VERSION>
Mahout	0.7_gphd_3_0_1_0	<PHD_MAHOUT_VERSION>
HCatalog	0.12.0_gphd_3_0_1_0	<PHD_HCATALOG_VERSION>
Sqoop	1.4.2_gphd_3_0_1_0	<PHD_SQOOP_VERSION>
Flume	1.4.0_gphd_3_0_1_0	<PHD_FLUME_VERSION>
Zookeeper	3.4.5_gphd_3_0_1_0	<PHD_ZOOKEEPER_VERSION>
Oozie	4.0.0_gphd_3_0_1_0	<PHD_OOZIE_VERSION>
bigtop-jsvc	1.0.15_gphd_3_0_1_0	<PHD_BIGTOP_JSVC_VERSION>
bigtop-utils	0.4.0_gphd_3_0_1_0	<PHD_BIGTOP_UTILS_VERSION>
bigtop-tomcat	6.0.36_gphd_3_0_1_0	<PHD_BIGTOP_TOMCAT_VERSION>

The following sections describe how to manually install Pivotal HD 2.0.

Prerequisites

- Oracle Java Development Kit (JDK) 1.7 must be installed on every machine before installing any of the Hadoop components.
- You must ensure that time synchronization and DNS are functioning correctly on all client and server machines. For example, run the following command to sync the time with NTP server:

```
$> service ntpd stop; ntpdate <DNS server IP address>; service ntpd start
```

Installation Notes

In this section we install packages by running the following command:

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

Where:

<version> is the PHD version

<nn> is the build number

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
- HDFS NFS Gateway Setup

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

hadoop-hdfs-secondarynamenode-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm

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

hadoop-hdfs-journalnode-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm

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

hadoop-hdfs-nfs3-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm

Type	Daemon
Requires	hadoop-hdfs
Description	Daemon scripts package for Hadoop NFS gateway, which provides a convenient method to manage NFS gateway start/stop as a Linux service.
Install on Nodes	Install on the node serving as as the NFS server

hadoop-hdfs-portmap-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm

Type	Daemon
Requires	hadoop-hdfs
Description	Daemon scripts package for Hadoop portmap, which provides a convenient method to manage portmap start/stop as a Linux service.
Install on Nodes	Install on the node serving as as the NFS server

```
hadoop-hdfs-zkfc-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm
```

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

```
hadoop-hdfs-fuse-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm
```

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

```
hadoop-libhdfs-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm
```

Type	Core
Requires	hadoop-hdfs
Description	Native implementation of the HDFS.
Install on Nodes	Install on servers that you want to run native hdfs.

```
hadoop-httpfs-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm
```

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

```
hadoop-doc-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm
```

Type	Doc
Requires	N/A
Description	Hadoop documentation package.

```
hadoop-client-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm
```

Type	Library
Requires	hadoop, hadoop-yarn, hadoop-mapreduce, hadoop-hdfs
Description	A set of symbolic link which gathers the libraries for programming Hadoop and submit Hadoop jobs.

Install on Nodes	Clients nodes that will be used to submit hadoop jobs.
-------------------------	--

Prerequisites: Core Package Setup

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.

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
```

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
```

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-hdfs-secondarynamenode-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm
```

HDFS NFS Gateway Setup

Install the Hadoop NFS gateway and portmap package on the workstation that will serve as HDFS NFS gateway and portmap:

```
$ 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-nfs3-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm  
$ sudo rpm -ivh working_dir/hadoop/rpm/hadoop-hdfs-portmap-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm
```

HDFS Configuration

HDFS configuration files are located in the following directory:

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

Refer to Apache Hadoop documentation for how to configure HDFS in distributed mode.

Usage

After installing the daemon package for Hadoop, you can start the daemons, as follows:

Starting HDFS

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

To start the Namenode daemon:

Format the Namenode before starting it, as follows:

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



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

Then start the Namenode by running the following commands:

Either:

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

Or:

```
$ sudo /etc/init.d/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 /etc/init.d/hadoop-hdfs-datanode start
```

To start the Secondary Namenode daemon:

Run either:

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

Or:

```
$ sudo /etc/init.d/hadoop-hdfs-secondarynamenode start
```

Starting NFS gateway

Three daemons are required to provide NFS service: portmap(or rpcbind), mountd and nfsd. The NFS gateway has both mountd and nfsd.

To start the portmap and NFS gateway daemon:

Run either:

```
$ sudo service rpcbind stop
$ sudo service hadoop-hdfs-portmap start
$ sudo service hadoop-hdfs-nfs3 start
```

Or:

```
$ sudo /etc/init.d/hadoop-hdfs-portmap start
$ sudo /etc/init.d/hadoop-hdfs-nfs3 start
```

To mount the export "/":

Make sure nfs-utils is installed on client:

```
$ sudo yum install -y nfs-utils
```

Then mount:

```
$ mount -t nfs -o vers=3,proto=tcp,nolock <nfsserver>:/ <mount_point>
```

Using HDFS

When the HDFS components are started, 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  
$ sudo -u hdfs hdfs dfs -copyFromLocal /etc/passwd /user/hadoop/
```



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

Stopping HDFS

Stop the Namenode Daemon:

Run either:

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

Or:

```
$ sudo /etc/init.d/hadoop-hdfs-namenode stop
```

Stop the Datanode Daemon:

Run either:

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

Or:

```
$ sudo /etc/init.d/hadoop-hdfs-datanode stop
```

Stop the Secondary Namenode Daemon:

Run either:

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

Or:

```
$ sudo /etc/init.d/hadoop-hdfs-secondarynamenode stop
```

Stop the NFS gateway Daemon:

Run either:

```
$ sudo service hadoop-hdfs-portmap stop  
$ sudo service hadoop-hdfs-nfs3 stop
```

Or:

```
$ sudo /etc/init.d/hadoop-hdfs-portmap stop  
$ sudo /etc/init.d/hadoop-hdfs-nfs3 stop
```

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

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

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

hadoop-yarn-nodemanager-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm	
--	--

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

hadoop-yarn-proxyserver-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm

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

hadoop-mapreduce-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm

Type	Core
Requires	hadoop-yarn
Description	Hadoop Mapreduce core libraries.
Install on Nodes	Install on all ResourceManager and NodeManager nodes.

hadoop-mapreduce-historyserver-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm

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

hadoop-doc-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm

Type	Doc
Requires	N/A
Description	Hadoop documentation package.

hadoop-client-<PHD_HADOOP_VERSION>-<nn>.x86_64.rpm

Type	Library
Requires	hadoop, hadoop-hdfs, hadoop-yarn, hadoop-mapreduce
Description	A set of symbolic link which gathers the libraries for programming Hadoop and submit Hadoop jobs.

Install on Nodes	Clients nodes that will be used to submit hadoop jobs.
-------------------------	--

Prerequisites: Core Package Setup

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.

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
```

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
```

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
```

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
```

YARN Configuration

Yarn configuration files are located in the following directory:

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

Refer to Apache Hadoop documentation for how to configure yarn in distributed mode.

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, create some working directories on HDFS, as follows:

Create working directories on HDFS:

The examples we show here are only examples; the exact steps depend upon your own environment and directory setup.

```
$ 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

The RM daemon only needs to be started on the master node.

To start RM, run either:

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

Or:

```
$ sudo /etc/init.d/hadoop-yarn-resourcemanager start
```

Once RM is started, you can visit its dashboard at: <http://localhost:8088/>

Starting NodeManager

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

To start NodeManager, run either:

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

Or:

```
$ sudo /etc/init.d/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.

To start the MapReduce History Server, run:

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

Or:

```
$ sudo /etc/init.d/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.



Before you start using YARN, make sure the 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 runs 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, for example the following command will print a list of available examples:

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

Stopping YARN

Stop the YARN daemons manually by running the following commands.

To stop the MapReduce HistoryServer Daemon:

Run:

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

Or:

```
$ sudo /etc/init.d/hadoop-mapreduce-historyserver stop
```

To stop the NodeManager Daemon:

Run:

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

Or:

```
$ sudo /etc/init.d/hadoop-yarn-nodemanager stop
```

To stop the ResourceManager Daemon:

Run:

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

Or:

```
$ sudo /etc/init.d/hadoop-yarn-resourcemanager stop
```

Hadoop Pseudo-distributed Configuration

This section provides instructions for installing Hadoop Pseudo-distributed Configuration.

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

Hadoop Pseudo-distributed Configuration Setup

Hadoop configuration files are located in the following directory:

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

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

Zookeeper

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

This section describes how to install Zookeeper.

Zookeeper RPM Packages

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

 Zookeeper doesn't require Hadoop Core Packages.

<code>zookeeper-<PHD_ZOOKEEPER_VERSION>-<nn>.noarch.rpm</code>	
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.

<code>zookeeper-server-<PHD_ZOOKEEPER_VERSION>-<nn>.noarch.rpm</code>	
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

<code>zookeeper-doc-<PHD_ZOOKEEPER_VERSION>-<nn>.noarch.rpm</code>	
Type	Documentation
Requires	N/A
Description	Zookeeper documentation package.

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.

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
```

Zookeeper Configuration

Zookeeper configuration files are located in the following directory

`/etc/gphd/zookeeper/conf`

This is the default configuration for quick reference and modification.

You can make modifications to these configuration templates or create your own configuration set.

Usage

Starting the Zookeeper Daemon

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

Start the daemons manually by running the following commands:

Either:

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

Or:

```
$ sudo /etc/init.d/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

Stop the Zookeeper server daemon manually by running the following commands:

Either:

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

Or:

```
sudo /etc/init.d/zookeeper-server stop
```

HBase

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

This section specifies how to install HBase.

Prerequisites

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

HBase RPM Packages

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

hbase-<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)
Requires	HBase Core Package
Description	Daemon scripts package to provide HBase service through thrift.

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

hbase-doc-<PHD_HBASE_VERSION>-<nn>.noarch.rpm	
Type	Documentation
Description	HBase documentation package.

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
```

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
```

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
```

HBase Thrift Server Setup

[OPTIONAL]

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

```
$ 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
```

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
```

HBase Configuration

HBase configuration files are located in the following directory:

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

This is the default configuration for quick reference and modification.

You can make modifications to these configuration templates or create your own configuration set.

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 the ownership for `hbase.rootdir`:

```
$ sudo -u hdfs hdfs dfs -chown hbase:hadoop /hbase
```

4. Add `hbase` user to the `hadoop` group if not already present, by running:

```
$ sudo usermod -G hadoop hbase
```

Usage

Starting the HBase Daemon

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

Start the daemons manually by running the following commands:

Run:

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

Or

```
$ sudo /etc/init.d/hbase-master start
```

Starting the HRegionServer daemon

Start the daemons manually by running the following commands:

Run:

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

Or

```
$ sudo /etc/init.d/hbase-regionserver start
```

Starting the Hbase Thrift server daemon

[OPTIONAL]

Start the daemons manually by running the following commands:

Run:

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

Or

```
$ sudo /etc/init.d/hbase-thrift start
```

Starting the Hbase Rest server daemon

[OPTIONAL]

Start the daemons manually by running the following commands:

Run:

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

Or

```
$ sudo /etc/init.d/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, 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

Stop the daemons manually by running the following commands:

Run:

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

Or

```
$ sudo /etc/init.d/hbase-master stop
```

Stopping the HRegionServer daemon

Stop the daemons manually by running the following commands:

Run:

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

Or

```
$ sudo /etc/init.d/hbase-regionserver stop
```

Stopping the Hbase Thrift server daemon

[OPTIONAL]

Stop the daemons manually by running the following commands:

Run:

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

Or

```
$ sudo /etc/init.d/hbase-thrift stop
```

Stopping the Hbase Rest server daemon

[OPTIONAL]

Stop the daemons manually by running the following commands:

Run:

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

Or

```
$ sudo /etc/init.d/hbase-rest stop
```

Hive

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

This section specifies how to install Hive.

Hive Components

A Hive installation consists of the following components:

- `hive-core`
- `hive-metastore`
- `hive-server`
- `hive-server2`

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:

- DB Server (we recommend using PostgreSQL)

Hive RPM Packages

Hive consists of hive core, hive metastore daemon, hive server and hive server2 packages:

<code>hive-<PHD_HIVE_VERSION>-<nn>.noarch.rpm</code>	
Type	Core
Requires	Hadoop, HBase Core Packages
Description	Hive core package provides the executables, libraries, configuration files and documentations.
Install on Nodes	Hive client & server workstation

<code>hive-server-<PHD_HIVE_VERSION>-<nn>.noarch.rpm</code>	
Type	Daemon (hive server)
Requires	Hive Core Package
Description	Daemon scripts package to provide Hive service through thrift
Install on Nodes	Hive 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 server2 node

Installing DB for Hive Metastore

Set up DB (Recommend using PostgreSQL) on the hive metastore Node

1. Install postgresql-server on hive metastore node as root:

```
$ sudo yum install postgresql-server
```

2. Initialize the database:

```
$ sudo service postgresql initdb
```

3. Open the /var/lib/pgsql/data/postgresql.conf file and set the following values:

```
listen_addresses = '*'
standard_conforming_strings = off
```

4. 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.

Then add the following lines:

```
local all all trust
host all all 0.0.0.0 0.0.0.0 trust
```

5. Create /etc/sysconfig/pgsql/postgresql file and add the following parameter/value pair:

```
PGPORT=10432
```

6. Start the database:

```
$ sudo service postgresql start
```

7. Create the user, database:

```
$ sudo -u postgres createuser -p 10432 -D -S -R -P hive
$ sudo -u postgres createdb -p 10432 -O hive metastore
```

8. Run postgres sql script to create hive schema in postgres db:

```
$ sudo -u postgres psql -p 10432 -d metastore -U hive -f
/usr/lib/gphd/hive/scripts/metastore/upgrade/postgres/hive-schema-<HIVE_VERSION>.postgres.sql
```

Hive MetaStore Server Setup

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

1. Install the Hive metastore:

```
$ 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.hwi.war.file</name>
    <value>/usr/lib/gphd/hive/lib/hive-hwi.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>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. Link the postgresql jar file:

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

4. Start the hive-metastore:

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

Hive Server Setup

[OPTIONAL]

Install the Hive core package and Hive server package to provide Hive service:

```
$ 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
```

Set up PostgreSQL jdbc driver

Copy the postgresql-jdbc.jar from the HIVE_METASTORE node to /usr/lib/gphd/hive/lib on the HIVE_SERVER node

Hive Server2 Setup

[OPTIONAL]

Install the Hive core package and Hive server2 package to provide Hive service:

```
$ 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
```

Open the /etc/gphd/hive/conf/hive-site.xml and add the following properties:

```

<property>
  <name>hive.server2.thrift.port</name>
  <value><CHANGE_TO_HIVE_SERVER2_PORT></value>
  <description>Port number of HiveServer2 Thrift interface.
  Can be overridden by setting $HIVE_SERVER2_THRIFT_PORT</description>
</property>
<property>
  <name>hive.server2.thrift.bind.host</name>
  <value><CHANGE_TO_HIVE_SERVER2_HOSTNAME></value>
  <description>Bind host on which to run the HiveServer2 Thrift interface.
  Can be overridden by setting $HIVE_SERVER2_THRIFT_BIND_HOST</description>
</property>

```

Set up PostgreSQL jdbc driver

Copy the `postgresql-jdbc.jar` from the `HIVE_METASTORE` node to `/usr/lib/gphd/hive/lib` on the `HIVE_SERVER2` node

Hive Configuration

The Hive configuration files are located in the following directory:

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

You can make modifications to this configuration template or create your own.

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 /user/hive/warehouse
```

3. Set permissions for the `hive.warehouse.dir`

```
$ sudo -u hdfs hadoop fs -chmod 775 /user/hive/warehouse
```

4. Set the ownership for the `hive.warehouse.dir`

```
$ sudo -u hdfs hadoop fs -chown hive:hadoop /user/hive/warehouse
```

5. Add hive user to hadoop group if not already present using:

```
$ sudo usermod -G hadoop hive
```


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
```

Set up PostgreSQL jdbc driver

Copy the `postgresql-jdbc.jar` from the `HIVE_METASTORE` node to `/usr/lib/gphd/hive/lib` on the `HIVE_CLIENT` node

Hive Usage

Start/Stop Hive Metastore Server

Start/stop the Hive Metastore server daemon by running:

Run:

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

Or

```
$ sudo /etc/init.d/hive-metastore start
$ sudo /etc/init.d/hive-metastore stop
```

Start/Stop Hive Server

[Optional]

Start/stop the Hive server daemon by running:

Run:

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

Or

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

Start/Stop Hive Server2

[Optional]

Start/stop Hive server2 daemon by running:

Run:

```
$ sudo service hive-server2 start
$ sudo service hive-server2 stop
```

Or

```
$ sudo /etc/init.d/hive-server2 start
$ sudo /etc/init.d/hive-server2 stop
```

Start HiveServer Client

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

```
$ hive
```

For example:

```
$ hive -e "CREATE TABLE test(id string, name string);"
$ hive -e "show tables"
OK
test
```

Start HiveServer2 Client

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

```
$ beeline
```

For example:

```
$ beeline
beeline> !connect jdbc:hive2://<hostname> <username> <password> org.apache.hive.jdbc.HiveDriver
0: jdbc:hive2://localhost> create table test1(id string, name string);
0: jdbc:hive2://localhost> show tables;
+-----+
| tab_name |
+-----+
| test     |
| test1    |
+-----+
```

Hcatalog

HCatalog is a metadata and table management system.

This section specifies how to install Hcatalog.

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.

Hcatalog RPM Packages

Hcatalog consists of one core package, a thrift sever daemon package that provides catalog service and a web server daemon package which provides catalog service through http.

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

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

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

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

Install on Nodes	Webhcat server node.
-------------------------	----------------------

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
```

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
```

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
```

Webhcat Server Setup

[OPTIONAL]

Install the Hcatalog core package, Webhcat core package and Webhcat daemon package to provide Webhcat 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
```

Hcatalog Configuration

The Hcatalog configuration files are located in the following directories:

/etc/gphd/hive/conf/

/etc/gphd/hcatalog/conf/

You can make modifications to these configuration templates or create your own.

Usage

Start/Stop Hcatalog Server

Start/stop Hcatalog server daemon by running the following commands:

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
```



hcatalog-server and hive-metastore server can not run on the same node at the same time. These 2 service should be put on different nodes.

Start/Stop Webhcat Server

Start/stop Webhcat server daemon by running the following commands:

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
```

Webhcat example

With this example, we use webhcat to create tables on hive:

```
# Create table use hive:
$ hive -e "CREATE TABLE test(id string, name string);"
# Get table by using webhcat, you need to change hostname and username to appropriate value
$ curl -s 'http://<hostname>:50111/templeton/v1/ddl/database/default/table/test?user.name=username'
{"columns":[{"name":"id","type":"string"}, {"name":"name","type":"string"}], "database":"default", "tabl
```

Start Hcatalog Client

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

```
$ hcat
$ cat test_data
key,value1
key,value2
key,value3
key,value4
$ hcat -e "CREATE TABLE hcat_test(key string, value string) ROW FORMAT DELIMITED FIELDS TERMINATED
BY ','"
$ hive -e "LOAD DATA LOCAL INPATH 'test_data' OVERWRITE INTO TABLE hcat_test"
$ hive -e "SELECT COUNT(*) FROM hcat_test"
4
```

Pig

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

This section specifies how to install Pig.

Prerequisites

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

Pig RPM Packages

Pig has only one core package.

pig-<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 package.

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
```

Pig Configuration

Pig configuration files are located in the following directory:

```
/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.

Usage

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

```
$ pig
```

Check the `pig` command usage by running:

```
$ pig -help
```

Piggybank Usage

Piggybank is a java library which includes a lot of useful Pig UDFs. Piggybank provides UDFs for different Pig storage functions, math functions, string functions and datetime functions, etc.

After you have installed Pig rpm package, piggybank library is also installed on the host.

The piggybank jar is in the following location:

```
/usr/lib/gphd/pig/piggybank.jar.
```

The library jars which piggybank depends on are in the following location:

```
/usr/lib/gphd/pig/lib/
```

Use the following script to register piggybank library in your pig script:

```
register /usr/lib/gphd/pig/lib/avro-*.jar
register /usr/lib/gphd/pig/lib/commons-*.jar
register /usr/lib/gphd/pig/lib/groovy-all-*.jar
register /usr/lib/gphd/pig/lib/guava-*.jar
register /usr/lib/gphd/pig/lib/jackson-*.jar
register /usr/lib/gphd/pig/lib/joda-time-*.jar
register /usr/lib/gphd/pig/lib/json-simple-*.jar
register /usr/lib/gphd/pig/lib/parquet-pig-bundle-*.jar
register /usr/lib/gphd/pig/lib/protobuf-java-*.jar
register /usr/lib/gphd/pig/lib/snappy-java-*.jar
register /usr/lib/gphd/pig/piggybank.jar
```

There are some notes for using Hive storage (such as `HiveColumnarStorage`) in piggybank.

- PHD hive must be installed. Please refer to [Hive section](#) for hive installation.
- You can register piggybank dependency jars as needed in your pig script with above code.
- Additionally, use the following pig code to register hive jars in your script

```
register /usr/lib/gphd/hive/hive-exec-*.jar
register /usr/lib/gphd/hive/hive-common-*.jar
```


Mahout

Mahout is a scalable machine learning and data mining library.

This section specifies how to install Mahout.

Prerequisites

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

Mahout RPM Packages

Mahout has only one core package.

mahout-<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 client workstation

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
```

Mahout Configuration

Mahout configuration files are located in the following directory:

```
/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.

Usage

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

```
$ mahout PROGRAM
```

Check the full list of mahout programs by running:

```
$ mahout
```

Flume

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

This section specifies how to install Flume.

Prerequisites

As Flume is built on top of Hadoop, the Hadoop package must be installed to get Flume running correctly. (Hadoop core and hadoop hdfs should be installed)

Flume RPM Packages

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

flume-<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.

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
```



User `flume` and group `flume` should be created with correct configuration, including `uid`, `gid`, `home_dir` and `shell`.

Check in the following paths: `/etc/passwd`, `/etc/group`

Flume Agent Setup

[Optional]

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

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

Flume Agent Configuration

Flume configuration files are located in the following directory:

`/etc/gphd/flume/conf/`

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

You can modify these configuration templates or create your own configuration set.

Usage

Starting Flume Client

Simple configuration example:

```
$ cat example.conf
al.sources = r1
al.sinks = k1
al.channels = c1
# Describe/configure the source
al.sources.r1.type = netcat
al.sources.r1.bind = localhost
al.sources.r1.port = 44444
# Describe the sink
al.sinks.k1.type = logger
# Use a channel which buffers events in memory
al.channels.c1.type = memory
al.channels.c1.capacity = 1000
al.channels.c1.transactionCapacity = 100
# Bind the source and sink to the channel
al.sources.r1.channels = c1
al.sinks.k1.channel = c1
```

Run Flume scripts on a client machine:

```
# terminal 1:

$ flume-ng agent --conf conf --conf-file example.conf --name a1 -Dflume.root.logger=INFO,console
14/03/19 06:00:42 INFO source.NetcatSource: Created
serverSocket:sun.nio.ch.ServerSocketChannelImpl[/127.0.0.1:44444]
14/03/19 06:01:18 INFO sink.LoggerSink: Event: { headers:{} body: 68 65 6C 6C 6F 0D
hello. }

# terminal 2:

$ telnet localhost 44444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello
OK
```

Starting/Stopping Flume Agent Server

Flume agent configuration example:

```
$ cat /etc/gphd/flume/conf/flume.conf
agent.sources = r1
agent.sinks = k1
agent.channels = c1

# Describe/configure the source
agent.sources.r1.type = netcat
agent.sources.r1.bind = localhost
agent.sources.r1.port = 44444

# Describe the sink
agent.sinks.k1.type = hdfs
agent.sinks.k1.hdfs.path = hdfs://localhost/user/flume/
agent.sinks.k1.hdfs.fileType = DataStream

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

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

Start/stop Flume agent server daemon by running the following commands:

Run:

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

Or

```
$ sudo /etc/init.d/flume-agent start
$ sudo /etc/init.d/flume-agent stop
$ sudo /etc/init.d/flume-agent status
```

Sqoop

Sqoop is a tool designed for efficiently transferring bulk data between [Apache Hadoop](#) and structured datastores such as relational databases. For more details, refer to the Apache Sqoop page:

<http://sqoop.apache.org/>

This section specifies how to install Sqoop.

Prerequisites

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

Sqoop RPM Packages

Sqoop 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

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
```

Sqoop Metastore Setup

[Optional]

Install the Sqoop core package and Sqoop metastore 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
```

Sqoop Metastore Configuration

The Sqoop metastore configuration file are in the following location:

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

These are the default configuration templates for quick reference and modification.

You can modify these configuration templates or create your own configuration set.

Usage

Starting/Stopping Sqoop Metastore Server

Start/stop Sqoop metastore server daemon by running the following commands:

Either:

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

Or

```
$ sudo /etc/init.d/sqoop-metastore start  
$ sudo /etc/init.d/sqoop-metastore stop  
$ sudo /etc/init.d/sqoop-metastore status
```

Starting Sqoop Client

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

```
$ sqoop
```

Check the sqoop command usage by running:

```
$ sqoop help
```

Sqoop Client Example

In this example, you are using sqoop to import a MySQL database table into HDFS.

To run this example, in addition to a correctly installed and configured PHD, you also need:

1. Install and run MySQL instance correctly (refer to MySQL's official document at <http://dev.mysql.com/doc/index-topic.html>).
2. Install MySQL official JDBC driver and copy `mysql-connector-java.jar` into `/usr/lib/gphd/sqoop/lib`.
3. Create MySQL database test and MySQL table student.

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

Then run sqoop to import the table to HDFS:

```
$ sqoop import --connect jdbc:mysql://<mysql server host>/test --table student --username
<username> --target-dir hdfs://<namenode host>/tmp/sqoop_output
```

Where:

`<mysql server host>` is the host name on which your MySQL instance is running

`<username>` is the user name of the user running this command

`<namenode host>` is the host name on which your name node is running.

Ensure the path you specified with `--target-dir` option in the above command above is a valid path on HDFS and the user account you are using has write permission on that location.

This should start a map/reduce job. Once it has finished, you should find files under the HDFS path you specified with `--target-dir`, and you should find the file content is the data of your MySQL table.

Oozie

Oozie is a workflow scheduler system to manage Apache Hadoop jobs.

This section specifies how to install Oozie.

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, Hive and Pig packages must be installed. See Hive section and Pig section for their installation instructions.

Oozie RPM Packages

Oozie contains a oozie-client rpm package and a oozie package. Oozie package depends on oozie-client package.

oozie-client-<PHD_OOZIE_VERSION>-<nn>.noarch.rpm	
Type	Client and Core
Requires	bigtop-util, hadoop-client
Description	Oozie client package provides oozie library and client binray to connect to Oozie service.
Install on Nodes	Oozie service node and Oozie client node

oozie-<PHD_OOZIE_VERSION>-<nn>.noarch.rpm	
Type	Daemon(Oozie server)
Requires	bigtop-tomcat, hadoop-client, oozie-client;
Description	Daemon package to provide Oozie service.
Install on Nodes	Oozie service node

Oozie client Setup

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

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



User "oozie" and group "oozie" are created with correct configuration (uid oozie, gid oozie). It is a non-login user.

Oozie server Setup [Optional]

Install the oozie-client package and oozie package to provide Oozie service:

```
$ sudo rpm -ivh working_dir/utility/rpm/bigtop-utils-<PHD_BIGTOP_UTILS_VERSION>-<nn>.noarch.rpm
$ sudo rpm -ivh working_dir/utility/rpm/bigtop-tomcat-<PHD_BIGTOP_TOMCAT_VERSION>-<nn>.noarch.rpm
$ 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
```

Oozie Configuration

Oozie configuration files are located in the following directory:

```
/etc/gphd/oozie/conf/
```

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

You can modify these configuration templates or create your own configuration set.

Oozie environment configuration

You can overwrite the oozie environment as long as exporting vars in

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

For example, if you want to define the oozie data directory, export OOZIE_DATA in

```
/etc/gphd/oozie/conf/oozie-env.sh:
```

```
export OOZIE_DATA=<YOUR_PATH>
```

Make sure that the owner and user group of <YOUR_PATH> is oozie:oozie.

Setup Database

OPTIONAL

By default, Oozie is configured to use Embedded Derby, however Oozie also works with HSQL, Derby, MySQL, Oracle and PostgreSQL databases.

Oozie bundles the JDBC drivers for HSQL, Embedded Derby and PostgreSQL.

HSQL is normally used for testcases as it is an in-memory database and all data is lost everytime Oozie is stopped.

If you are using MySQL, Oracle or PostgreSQL, the Oozie database schema must be created. By default, Oozie creates its tables automatically.

The `bin/addtowar.sh` and the `oozie-setup.sh` scripts have an option `-jars` that can be used to add the Oracle or MySQL JDBC driver JARs to the Oozie WAR file.

The SQL database used by Oozie is configured using the following configuration properties (default values shown):

```
oozie.db.schema.name=oozie
oozie.service.JPAService.create.db.schema=true
oozie.service.JPAService.validate.db.connection=false
oozie.service.JPAService.jdbc.driver=org.apache.derby.jdbc.EmbeddedDriver
oozie.service.JPAService.jdbc.url=jdbc:derby:${oozie.data.dir}/${oozie.db.schema}

oozie.service.JPAService.jdbc.username=sa
oozie.service.JPAService.jdbc.password=
oozie.service.JPAService.pool.max.active.conn=10
```

These values should be changed to match the database you are using.



- If the `oozie.db.schema.create` property is set to `true` (default) the Oozie tables are created automatically if they are not found in the database at Oozie start-up time. In a production system this option should be set to `false` once the database tables have been created.
- If the `oozie.db.schema.create` property is set to `true`, the `oozie.service.JPAService.validate.db.connection` property value is ignored and Oozie handles it as set to `false`.

Usage

Oozie 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]
```

Check the oozie command usage by running:

```
$ oozie help
```

Initialize Oozie server

[Optional]

Before starting Oozie service, follow the steps below to initialize Oozie server.

1. Add the following configuration to hadoop configuration `core-site.xml`. Then 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 /user/oozie
```

3. Create the oozie database:

```
$ sudo service oozie init
```

4. Download `extjs-2.2` from here <http://extjs.com/deploy/ext-2.2.zip>. Put zip file in a new directory named `/tmp/oozie-libext`.

```
$ wget http://extjs.com/deploy/ext-2.2.zip
$ mkdir -p /tmp/oozie-libext
$ mv ext-2.2.zip /tmp/oozie-libext
```

5. Setup oozie tomcat war file:

```
$ sudo -u oozie oozie-setup prepare-war -d /tmp/oozie-libext/
```

6. 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]

Start/stop Oozie server by running the following commands:

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
```

Submit Oozie example workflows

1. Expand the examples:

```
$ mkdir /tmp/oozie-example
$ cd /tmp/oozie-example
$ tar xzf /usr/lib/gphd/oozie/oozie-examples.tar.gz
```

2. Change the job properties in the examples.

Change the following files:

```
/tmp/oozie-example/examples/apps/map-reduce/job.properties
/tmp/oozie-example/examples/apps/hive/job.properties
/tmp/oozie-example/examples/apps/pig/job.properties
```

In each file, set the following properties:

```
nameNode=hdfs://<namenode-host>:<namenode-port>
jobTracker=<resource-manager-host>:<resource-manager-port>
```

Use the exact hostname and service port in your cluster.

3. Edit the Oozie `workflow.xml` as follows:

The Oozie `workflow.xml` is in the following directory:

```
/tmp/oozie-example/examples/apps/hive
```

Add the NameNode variable as a prefix to all paths, for example:

```
<param>INPUT=${nameNode}/user/${wf:user()}/${examplesRoot}/input-data/table</param>
<param>OUTPUT=${nameNode}/user/${wf:user()}/${examplesRoot}/output-data/hive</param>
```

Also make sure to reference the `hive-oozie-site.xml` using the `job-xml` tag in the workflow. The `<job-xml>` element needs to be put inside the `<hive>` element between the `<prepare>` and `<configuration>` elements in the `examples/apps/hive/workflow.xml` file, as shown below:

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="hive-wf">
  <start to="hive-node"/>
  <action name="hive-node">
    <hive xmlns="uri:oozie:hive-action:0.2">
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <prepare>
        <delete path="${nameNode}/user/${wf:user()}/${examplesRoot}/output-data/hive"/>
        <mkdir path="${nameNode}/user/${wf:user()}/${examplesRoot}/output-data"/>
      </prepare>
      <job-xml>${nameNode}/user/oozie/hive-oozie-site.xml</job-xml>
      <configuration>
        <property>
          <name>mapred.job.queue.name</name>
          <value>${queueName}</value>
        </property>
      </configuration>
      <script>script.q</script>
      <param>INPUT=${nameNode}/user/${wf:user()}/${examplesRoot}/input-data/table</param>
      <param>OUTPUT=${nameNode}/user/${wf:user()}/${examplesRoot}/output-data/hive</param>
    </hive>
    <ok to="end"/>
    <error to="fail"/>
  </action>
  <kill name="fail">
    <message>Hive failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>
  </kill>
  <end name="end"/>
</workflow-app>
```

4. Put example code onto HDFS:

```
$ hdfs dfs -put examples /user/<username>
```

Where <username> is the name of user who issues this command.

5. Submit map reduce example workflow

a. Submit workflow:

```
$ oozie job -oozie http://localhost:11000/oozie -config
examples/apps/map-reduce/job.properties -run
job: <oozie-job-id>
```

b. Check workflow status.

Where <oozie-job-id> is the same id in the output of the last command.

```
$ oozie job -oozie http://localhost:11000/oozie -info <oozie-job-id>
```

6. Oozie Setup for Hive:

a. Remote Metastore Mode (recommended):

Put the Hive jars into the Tomcat class loader path. Make the following change in the `/var/lib/gphd/oozie/tomcat-deployment/conf/catalina.properties` file:

```
common.loader=/var/lib/gphd/oozie/*.jar,/usr/lib/gphd/hadoop/client/*.jar,/usr/lib/gphd/hive/li
```

b. Local Metastore Mode:

Upload the JDBC driver to Oozie sharelib

To enable the local metastore mode, comment out the `hive.metastore.uris` property and verify that Hive still works properly at the command-line. In local metastore mode, Oozie hive actions do not connect to the Hive Metastore, but instead talk to the database directly. In this setup, the appropriate JDBC driver (for example, for Postgres) needs to be made available to hive jobs running within Oozie:

```
hdfs dfs -put /usr/lib/gphd/hive/lib/postgresql-jdbc.jar /user/oozie/share/lib/hive
```

7. Submit hive example workflow.

a. Upload hive configuration file onto HDFS

```
$ sudo -u oozie hdfs dfs -put /etc/gphd/hive/conf/hive-site.xml  
/user/oozie/hive-oozie-site.xml
```



When uploading hive configuration file to HDFS, do not use `hive-site.xml` as the file name. This is because hive action in Oozie overwrites `hive-site.xml` file.

In the Oozie workflow file, use

`<job-xml>${nameNode}/user/oozie/hive-oozie-site.xml</job-xml>` to refer to the hive configuration file.

b. Submit workflow:

```
$ oozie job -oozie http://localhost:11000/oozie -config examples/apps/hive/job.properties  
-run  
job: <oozie-job-id>
```

c. Check workflow status.

Where `<oozie-job-id>` is the same id in the output of last command.

```
$ oozie job -oozie http://localhost:11000/oozie -info <oozie-job-id>
```

8. Submit pig example workflow.

a. submit workflow:

```
$ oozie job -oozie http://localhost:11000/oozie -config examples/apps/pig/job.properties
-run
job: <oozie-job-id>
```

b. Check workflow status.

Where <oozie-job-id> is the same id in the output of last command.

```
$ oozie job -oozie http://localhost:11000/oozie -info <oozie-job-id>
```

Oozie in HA Mode - Best Practices

- Ensure that HA is configured correctly and identically on all nodes, including client nodes. Specifically, ensure that the following variables are set appropriately in `hdfs-site.xml`:

```
dfs.nameservices
```

```
dfs.ha.namenodes.nameservice ID
```

```
dfs.namenode.rpc-address.nameservice ID.name node ID
```

```
dfs.namenode.http-address.nameservice ID.name node ID
```

```
dfs.namenode.shared.edits.dir
```

```
dfs.client.failover.proxy.provider.nameservice ID
```

```
dfs.ha.fencing.methods
```

And in `core-site.xml`:

```
fs.defaultFS
```

- Use the namenode HA service in `mapreduce.job.hdfs-servers` in `yarn-site.xml`:

```
<property>
  <name>mapreduce.job.hdfs-servers</name>
  <value>hdfs://test</value>
</property>
```

- While using Namenode HA, create all tables using the HA service as the HDFS location:

```
CREATE EXTERNAL TABLE test (a INT) STORED AS TEXTFILE LOCATION
'hdfs://test/user/myuser/examples/input-data/table/';
```

- Verify that all tables in Hive are created using the HA service as the HDFS location (note the location in the example below refers to `hdfs://test/`, which is the HA service.)

```
hive> describe extended mytable;
OK
a                                int                                None

Detailed
Table Information      Table(tableName:mytable, dbName:default,
owner:gpadmin@PIVOTAL, createTime:1391839636, lastAccessTime:0,
retention:0, sd:StorageDescriptor(cols:[FieldSchema(name:a, type:int, comment:null)],
location:hdfs://test/user/gpadmin/examples/input-data/mytable,
inputFormat:org.apache.hadoop.mapred.TextInputFormat,
outputFormat:org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat,
compressed:false,
numBuckets:-1, serdeInfo:SerDeInfo(name:null,
serializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe,
parameters:{serialization.format=1}),
bucketCols:[], sortCols:[], parameters:{},
skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[],
skewedColValueLocationMaps:{}), storedAsSubDirectories:false),
partitionKeys:[], parameters:{EXTERNAL=TRUE,
transient_lastDdlTime=1391839636}, viewOriginalText:null,
viewExpandedText:null, tableType:EXTERNAL_TABLE)
```

Hamster

Hamster is a framework that enables users running MPI programs on Apache Hadoop YARN platform.

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

Prerequisites

Hamster is dependent upon Hadoop (Apache Hadoop 2.2.0 / PHD 2.0), Hamster itself contains three parts, hamster-core (client and application master library), hamster-rte (plugins for OpenMPI), OpenMPI.

Hamster RPM Packages

Hamster contains hamster-core package, hamster-rte package and openmpi package.

hamster-core-<PHD_HAMSTER-CORE_VERSION>-1.x86_64.rpm	
Type	Client and application master library
Requires	
Description	Hamster-core installation package
Install on Nodes	Every node in the YARN cluster and the client workstation that will access the YARN service.

hamster-rte-<PHD_HAMSTER-RTE_VERSION>-1.x86_64.rpm	
Type	Plugins for OpenMPI
Requires	
Description	Hamster runtime environment installation package
Install on Nodes	Every node in the YARN cluster and the client workstation that will access the YARN service.

openmpi-<PHD_OPENMPI_VERSION>-1.x86_64.rpm	
Type	OpenMPI
Requires	
Description	OpenMPI installation package
Install on Nodes	Every node in the YARN cluster and the client workstation that will access the YARN service.

Hamster-core Setup

Install the hamster-core package on the nodes in the YARN cluster and the client workstation that will access the YARN service:

```
$ sudo rpm -ivh working_dir/hamster/rpm/hamster-core-<PHD_HAMSTER-CORE_VERSION>-1.x86_64.rpm
```

OpenMPI Setup

Install the OpenMPI package on the nodes in the YARN cluster and the client workstation that will access the YARN service:

```
$ sudo rpm -ivh working_dir/hamster/rpm/openmpi-<PHD_OPENMPI_VERSION>-1.x86_64.rpm
```

Hamster-rte Setup

Install the hamster-rte package on the nodes in the YARN cluster and the client workstation that will access the YARN service:

```
$ sudo rpm -ivh working_dir/hamster/rpm/hamster-rte-<PHD_HAMSTER-RTE_VERSION>-1.x86_64.rpm
```

Running an example

Run the following example to check that Hamster is working:

1. Use the `hello_c` program in OpenMPI's examples folder to validate hamster, this file needs be placed in all nodes with same directory.
2. Execute the following command:

```
$ hamster -np 3 /absolute/path/to/hello_c
```

The result should look like this:

```
13/08/26 07:36:14 INFO cli.HamsterCli: tracking URL is:
http://gphd-vm33:8088/proxy/application_1377213904799_0100/
13/08/26 07:36:16 INFO cli.HamsterCli: yarn application state transfered from [ACCEPTED] to
[RUNNING]
13/08/26 07:36:19 INFO cli.HamsterCli: yarn application state transfered from [RUNNING] to
[FINISHED]
13/08/26 07:36:19 INFO cli.HamsterCli: AppMaster is successfully finished.
```

This created an application on YARN, and it will negotiate with the YARN resource manager and execute MPI processes in allocated resources.

You can copy the tracking URL to your browser, and you can checkout your processes' realtime outputs. Note that this page only exists when the job is running.

Currently, MPI processes in Hamster will redirect their output to files in the Hadoop log folder, you can access them in `yarn.nodemanager.log-dirs` in `yarn-site.xml` under `$HADOOP_CONF_DIR`.

Hamster Usage

Display Hamster command usage by running:

```
$ hamster --help
```

Most of the original OpenMPI command line options are supported by hamster, including:

Option	Description
-c, -n, --n, -np, --np, -max-vm-size, --max-vm-size	Number of processes to run
-bynode, --bynode	Whether to map and rank processes round-robin by node
-byslot, --byslot	Whether to map and rank processes round-robin by slot
-display-allocation, --display-allocation	Display the allocation being used by this job
-display-devel-allocation, --display-devel-allocation	Display a detailed list (mostly intended for developers) of the allocation being used by this job
-display-devel-map, --display-devel-map	Display a detailed process map (mostly intended for developers) just before launch
-display-diffable-map, --display-diffable-map	Display a diffable process map (mostly intended for developers) just before launch
-display-map, --display-map	Display the process map just before launch
-display-topo, --display-topo	Display the topology as part of the process map (mostly intended for developers) just before launch
-h, --help	Show help message Preload the comma separated list of files to the remote machines current working directory before starting the remote process
-path, --path	PATH to be used to look for executables to start processes
-prefix, --prefix	Prefix where Open MPI is installed on remote nodes
-preload-files, --preload-files, -s, --preload-binary	Preload the comma separated list of files to the remote machines current working directory before starting the remote process.
-q, --quiet	Suppress helpful messages
-report-pid, --report-pid	Printout pid on stdout [-], stderr [+], or a file [anything else]
-report-uri, --report-uri	Printout URI on stdout [-], stderr [+], or a file [anything else]
-v, --verbose	Be verbose
-V, --version	Print version and exit
-x	

Option	Description
	Export an environment variable, optionally specifying a value (for example, <code>\-x foo\</code> exports the environment variable <code>foo</code> and takes its value from the current environment; <code>\-x foo=bar\</code> exports the environment variable name <code>foo</code> and sets its value to <code>\bar\</code> in the started processes)

These are some new command line options for hamster:

Option	Description
<code>-cpu, --cpu-per-proc</code>	Specify how many v-cores allocated to each MPI proc
<code>-mem, --mem-per-proc</code>	Specify how many memory (in MB) allocated to each MPI proc
<code>-max-at,</code> <code>--max-allocation-time</code>	Maximum time used do allocation (in milli-seconds), after timeout, all allocated containers will be returned and job will be failed, we use <code>min { \$max-alloc, yarn.resourcemanager.rm.container-allocation.expiry-interval-ms }</code> as the actual expired timeout
<code>-p, --policy</code>	Policy for scheduling. Valid value: { <code>default</code> , <code>compute-locality/cl</code> }
<code>-preload-archives,</code> <code>--preload-archives</code>	Preload the comma separated list of archives to the remote machines current working directory and un-zip before starting the remote process.

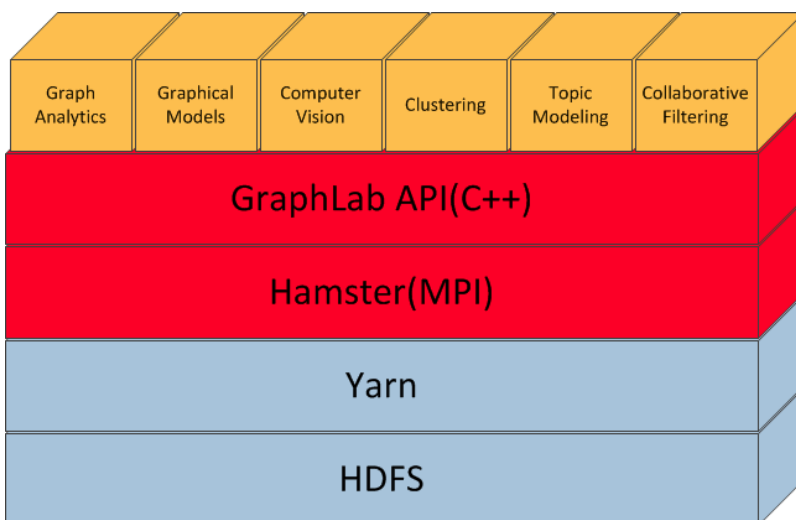
GraphLab

GraphLab is a powerful new system for designing and implementing parallel algorithms in machine learning. It is a graph-based, high performance, distributed computation framework written in C++. It makes use of MPI and has its own programming model.

You can find more information about GraphLab here: GraphLab.org.

Because GraphLab itself cannot run on YARN, we have integrated GraphLab to Hadoop using Hamster.

Following is an architecture diagram for GraphLab on Hadoop:



Prerequisites

- `hadoop-libhdfs*.rpm` and JRE package

If GraphLab algorithm's input/output are from/to HDFS, then `hadoop-libhdfs*.rpm` and JRE package are required.

When installing PHD2.0, make sure that `hadoop-libhdfs*.rpm` is installed and `$JAVA_HOME/jre/lib/amd64/server/libjvm.so` is found.

- Hamster

GraphLab RPM Packages

GraphLab consists of the following graphlab rpm package:

<code>graphlab-<PHD_GraphLab_VERSION>-1.x86_64.rpm</code>	
Type	GraphLab
Requires	Hamster, hadoop-libhdfs*.rpm, jre package

Description	GraphLab installation package
Install on Nodes	Every node in the YARN cluster and the client workstation that will access the YARN service.

GraphLab Setup

```
$ sudo rpm -ivh working_dir/graphlab/rpm/graphlab-<PHD_GraphLab_VERSION>-1.x86_64.rpm
```

Running an example

Following is an example to test that GraphLab is working:

Create a `connected_component.ini` file that contains the following:

```
1 2 4.0
2 3 1.0
3 4 5.0
4 5 2.0
5 3 3.0
```

Run the following commands to place this file in HDFS, and create a folder in HDFS for write output:

```
sudo -u hdfs hadoop fs -mkdir /graphlab
sudo -u hdfs hadoop fs -copyFromLocal /path/to/connected_component.ini
/graphlab/connected_component.ini
sudo -u hdfs hadoop fs -mkdir -p /graphlab/connected_component/output
sudo -u hdfs hadoop fs -chmod 777 /graphlab/connected_component/output
```

Run the following command to execute a GraphLab job:

```
hamster -np 2 /${graphlab_installation_dir}/graph_analytics/connected_component
--saveprefix=hdfs://${hdfs_hostname}:${hdfs_port}/graphlab/connected_component/output/
--graph=hdfs://${hdfs_hostname}:${hdfs_port}/graphlab/connected_component.ini --format=tsv
```

You should see the following log:

```
<...Trimmed...>
14/01/27 10:13:12 INFO cli.HamsterCli: tracking URL is: http:// <...Trimmed...>
14/01/27 10:13:15 INFO cli.HamsterCli: yarn application state transfered from [ACCEPTED] to
[RUNNING]
14/01/27 10:13:22 INFO cli.HamsterCli: yarn application state transfered from [RUNNING] to
[FINISHED]
14/01/27 10:13:22 INFO cli.HamsterCli: AppMaster is successfully finished
```

Usage

You can find Usage information for GraphLab's toolkits here: <http://docs.graphlab.org/toolkits.html>.

Note that here we are using Hamster as MPI runtime behind GraphLab. Therefore you should use `hamster` instead of `mpirun` or `mpiexec` to submit jobs to the cluster.

To display `hamster` command usage, run `hamster -h` or refer to [hamster usage](#) in this documentation.

YARN configurations for better using GraphLab on Hamster

Resource allocation / monitoring

In YARN, both virtual and physical memory usage for processes running on node managers are checked, if your program used more memory than requested, it will be killed by node manager. You can change your memory/cpu limit for your processes by specifying `-mem` (`--mem-per-proc`) and `-cpu` (`--cpu-per-proc`). For example, if you want your program using 2G memory and 2 cores, you can use following command to execute your job.

```
hamster -mem 2048 -cpu 2 -np N <your-program and parameters>
```

By default, `-mem` is 1024 (in MB) and `-cpu` is 1. But this is not enough, YARN itself has some properties will take effect on resource allocation and monitoring, all in `yarn-site.xml` in `$HADOOP_CONF_DIR`.

- **yarn.scheduler.maximum-allocation-mb and yarn.scheduler.maximum-allocation-vcores**

The two options will limit YARN's maximum resource allocation for each launched process. By default, they're 8192 (8GB memory) and 32 (cores), if your process need more than this limit, you need increase theirs values. GraphLab will consume lots of memory, **we suggest at least, specify maximum allocation memory in YARN to 16GB**. Otherwise, you may find your program frequently killed by node managers, which can be found in logs of node managers.

- **yarn.nodemanager.vmem-check-enabled and yarn.nodemanager.vmem-pmem-ratio**

First option indicated if we need check virtual memory. For example, some program will use a lot of virtual memory but barely use physic memory (GraphLab is an example). If you think it's unnecessary to check virtual memory, you can set it to **false** to disable this checking (default is true). The second option is the ratio for physical memory to virtual memory, default is 2.1. For example, if you specified 2G memory (which can be specified by `hamster -mem 2048 ...`) for your processes, the limit of its physical memory is 2G, and the limit of its virtual memory is $2.1 * 2G = 4.2G$. This process will be killed **either** it used 2G physical memory or 4.2G virtual memory. We suggest set a higher value of this to reduce your processes killed by node manager when it used too much virtual memory.

Fetch log for Hamster job

A traditional OpenMPI program has a feature that allows you to get logs when jobs are running. In Hamster 1.0 we don't support this because we launch MPI processes in a different way, however you can still get all log files after the job is finished.

We recommend that you set `yarn.log-aggregation-enable` to `true` in the `yarn-site.xml` (by default this is disabled). If this is left disabled, fetching logs for Hamster jobs is more complicated as you have to use the YARN web server to get your logs like other applications (for example, map-reduce).

When the `yarn.log-aggregation-enable` set to `true`, you need to be aware that the parent directory of `yarn.nodemanager.remote-app-log-dir` in HDFS should have write permission for the `yarnuser`. By default the `yarn.nodemanager.remote-app-log-dir` is set to `/tmp/logs`.

Once you have set the above properties, run the following command to get the log after your job has finished:

```
yarn logs -applicationId <application ID> -appOwner <Application Owner>
```

Note that `-appOwner` should be set to the user name used to submit Hamster job when you used a different user name execute the `yarn logs` command.

TroubleShooting

Q: When I run `rpm -ivh hamster-rte*.rpm`, the following error is thrown:

```
error: Failed dependencies: pkgconfig(zlib) is needed by avro-1.7.5-1.x86_64
```

A: You need to run `yum install zlib-devel.x86_64 zlib.x86_64` first

Q: I'm unable to see hadoop logs.

A: You need to add the following property in `yarn-site.xml`:

```
<property>
  <description></description>
  <name>yarn.nodemanager.delete.debug-delay-sec</name>
  <value>1800</value>
</property>
```

Then restart resource manager and nodemanager.

Q: When I run `hamster -np 3 /absolute/path/to/hello_c`, the following error is thrown:

```
Caused by:
org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.security.AccessControlException):
Permission denied: user=root, access=WRITE, inode="/user":hdfs:supergroup:drwxr-xr-x
at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecker.java:234)
at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecker.java:214)
at
org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.checkPermission(FSPermissionChecker.java:1
org.apache.hadoop.hdfs.server.namenode.FSNamesystem.checkPermission(FSNamesystem.java:5380)
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.checkPermission(FSNamesystem.java:5362)
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.checkAncestorAccess(FSNamesystem.java:5336)
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirsInternal(FSNamesystem.java:3583)
```

```
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirsInt(FSNamesystem.java:3553)
at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirs(FSNamesystem.java:3525)
at org.apache.hadoop.hdfs.server.namenode.NameNodeRpcServer.mkdirs(NameNodeRpcServer.java:745)
```

A: Run the following command:

```
sudo -u hdfs hadoop fs -chmod 777 /userhadoop fs -mkdir /user/root
```

Q: I see the following in the YARN logs:

```
14/02/17 09:52:30 ERROR event.HamsterEventHandler: exception when launch HNP process
java.io.IOException: Cannot run program "mpirun": error=13, Permission denied
at java.lang.ProcessBuilder.start(ProcessBuilder.java:1029)
at java.lang.Runtime.exec(Runtime.java:615)
at java.lang.Runtime.exec(Runtime.java:526)
at com.pivotal.hamster.appmaster.hnp.DefaultHnpLauncher$1.run(DefaultHnpLauncher.java:84)
at java.lang.Thread.run(Thread.java:722)
Caused by: java.io.IOException: error=13, Permission denied
at java.lang.UNIXProcess.forkAndExec(Native Method)
at java.lang.UNIXProcess.<init>(UNIXProcess.java:135)
at java.lang.ProcessImpl.start(ProcessImpl.java:130)
at java.lang.ProcessBuilder.start(ProcessBuilder.java:1021)
... 4 more
```

A: Check if the node which is running the hamster application has openmpi installed.

Q: I see the following information in logs:

```
LogType: stderr
LogLength: 48
Log Contents:
/bin/bash: /bin/java: No such file or directory
LogType: stdout
LogLength: 0
Log Contents:
```

A: Make sure \$JAVA_HOME is set:

Run this command: `echo $JAVA_HOME`, you should see something like this:

```
/usr/java/latest
```

Q: I see the following information in logs:

```
ERROR: fiber_control.cpp(launch:229): Check failed: b<nworkers [1 < 1]
```

A: Find another machine which has more than 1 core, this is a limitation.

Chapter 3 Manually Upgrading Pivotal HD Stack from 1.1.1 to 2.0

This section describes how to manually upgrade your Pivotal HD stack components from 1.1.1 to 2.0.

Topics:

- General Upgrade Tasks for Each Component
- Accessing the PHD 2.0 RPM
- PHD RPM Upgrade Limitations from PHD 1.1.1 to PHD 2.0
- Components to be Upgraded (ICM supported)
 - Upgrade Bigtop Utilities
 - Upgrade Zookeeper
 - Upgrade Hadoop
 - Upgrade HBase
 - Upgrade Hive
 - Upgrade Pig
 - Upgrade Mahout
- Components to be Upgraded (ICM not supported)
 - Upgrade Flume
 - Upgrade Sqoop
 - Upgrade Oozie
 - Upgrade HCatalog

PHD supports only stack-level upgrade; it doesn't support component-level upgrade. All PHD components should come from same release package and all PHD components need to upgrade to same release version.

General Upgrade Tasks for Each Component

For each component upgrade, the following steps need to be followed:

- Backup Configuration
- Stop Service
- RPM upgrade
- Restore Configuration
- Start Service

Accessing the PHD 2.0 RPM

Download and extract PHD package to working directory:

```
$ tar zxvf PHD-2.0.0.0-<nn>.tar.gz
$ ls -p PHD-2.0.0.0-<nn>
flume/      hadoop/    hbase/     hive/      oozie/           pig/      utility/
graphlab/   hamster/   hcatalog/  mahout/    open_source_licenses_PHD.txt  sqoop/    zookeeper/
```

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

Component	PHD Version	Replaced String
Hadoop	2.2.0_gphd_3_0_0_0	<PHD_HADOOP_VERSION>
HBase	0.96.0_gphd_3_0_0_0	<PHD_HBASE_VERSION>
Hive	0.12.0_gphd_3_0_0_0	<PHD_HIVE_VERSION>
Pig	0.12.0_gphd_3_0_0_0	<PHD_PIG_VERSION>
Mahout	0.7_gphd_3_0_0_0	<PHD_MAHOUT_VERSION>
HCatalog	0.12.0_gphd_3_0_0_0	<PHD_HCATALOG_VERSION>
Sqoop	1.4.2_gphd_3_0_0_0	<PHD_SQOOP_VERSION>
Flume	1.4.0_gphd_3_0_0_0	<PHD_FLUME_VERSION>
Zookeeper	3.4.5_gphd_3_0_0_0	<PHD_ZOOKEEPER_VERSION>
Oozie	4.0.0_gphd_3_0_0_0	<PHD_OOZIE_VERSION>
bigtop-jsvc	1.0.15_gphd_3_0_0_0	<PHD_BIGTOP_JSVCS_VERSION>
bigtop-utils	0.4.0_gphd_3_0_0_0	<PHD_BIGTOP_UTILS_VERSION>

PHD RPM Upgrade Limitations from PHD 1.1.1 to PHD 2.0

Some files in PHD 1.1.1 cannot be uninstalled after upgrade from PHD 1.1.1 to PHD 2.0.

Components to be Upgraded (ICM supported)

- Zookeeper
- Bigtop utilities
- HDFS
- Yarn
- HBase
- Hive
- Pig
- Mahout

The above components upgrade is supported through the CLI. Manual RPM upgrade is risky and upgrade through the CLI is strongly recommended.

Upgrade Bigtop Utilities

RPM Upgrade Bigtop Utilities

Upgrade Zookeeper

1. Backup existing configuration files.
2. Stop the zookeeper-server service.
3. RPM Upgrade Zookeeper.
4. Restore backup configuration files and make sure configuration is compatible with new installed version.
5. Restart the zookeeper-server service on Zookeeper server.

Upgrade Hadoop

Refer to http://wiki.apache.org/hadoop/Hadoop_Upgrade for hadoop data upgrade/migration. The following steps only cover hadoop configuration and packages upgrade.

1. Backup existing configuration files.
2. Stop Hadoop services.
3. RPM upgrade hadoop.

4. Restore backup configuration files and make sure configuration is compatible with new installed version.
5. Restart Hadoop services

Upgrade HBase

Refer to <http://hbase.apache.org/upgrading.html> for hbase data upgrade/migration. The following steps only cover hbase configuration and packages upgrade.

1. Backup existing configuration files.
2. Stop HBase master/region/thrift/restful service.
3. RPM upgrade Hbase.
4. Restore backup configuration files and make sure configuration is compatible with new installed version.
5. Restart HBase master/region/thrift/restful service.

Upgrade Hive

Refer to <https://cwiki.apache.org/confluence/display/Hive/Hive+Schema+Tool> for hive data upgrade/migration. The following steps only cover hive configuration and packages upgrade.

1. Backup existing configuration files.
2. Stop Hive Server, Hive Server2, metastore services.
3. RPM upgrade Hive
4. Restore backup configuration files and make sure configuration is compatible with new installed version.
5. If you use postgresql as the metastore DB, please copy postgresql-jdbc.jar to /usr/lib/gphd/hive/lib on hive-server node.
6. Restart Hive Server, Hive Server2, metastore services.

Upgrade Pig

1. Backup existing configuration files.
2. RPM upgrade Pig
3. Restore backup configuration files and make sure configuration is compatible with new installed version.
4. Restore any customized UDF jars to lib directory.

Upgrade Mahout

1. Backup existing configuration files.
2. RPM upgrade Mahout
3. Restore backup configuration files and make sure configuration is compatible with new installed version.

Components to be Upgraded (ICM not supported)

- Sqoop
- Flume
- Oozie
- Hcatlog

These instructions assume that the following components were already upgraded to PHD stack 1.1.1 and worked correctly:

- Zookeeper
- Hadoop
- Hbase
- Hive
- Pig
- Mahout

RPM upgrade requires `root` privileges on the nodes to be upgraded. Unless specified, all the following operations should be run by user `root`.

Upgrade Flume

1. Backup existing configuration files

```
$ cp -r /etc/gphd/flume/conf <where to save Flume configuration>
$ cp -r /etc/default/flume* <where to save Flume environment>
```

2. Stop Flume Agent Service

Before you begin the upgrade, stop the `flume-agent` service:

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

3. Upgrade the Flume Client

```
$ sudo rpm -U flume-<PHD_FLUME_VERSION>.noarch.rpm
$ sudo rpm -qa | grep flume
flume-<PHD_FLUME_VERSION>.noarch
```

4. Upgrade the Flume Agent

```
$ sudo rpm -U flume-<PHD_FLUME_VERSION>.noarch.rpm flume-agent-<PHD_FLUME_VERSION>.noarch.rpm
$ sudo rpm -qa | grep flume
flume-<PHD_FLUME_VERSION>.noarch
flume-agent-<PHD_FLUME_VERSION>.noarch
```

5. 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.

You also need to restore flume environment files in `/etc/default` if they were changed.

6. Restart Flume Agent Service

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

Upgrade Sqoop

1. Backup existing configuration files

We also require you back up the external jar files used to connect with database for Sqoop:

```
$ cp -r /etc/gphd/sqoop/conf <where to save Sqoop configuration>
$ cp -r /usr/lib/gphd/sqoop/lib <where to save external jar files>
$ cp -r /etc/default/sqoop* <where to save Sqoop environment>
```

2. Stop Sqoop Metastore Service

Before you run the upgrade, first stop the `sqoop-metastore` service:

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

3. Upgrade Sqoop Client

```
$ rpm -U sqoop-<PHD_SQOOP_VERSION>.noarch.rpm
$ rpm -qa | grep sqoop
sqoop-<PHD_SQOOP_VERSION>.noarch
```

4. Upgrade Sqoop Metastore

```
$ rpm -U sqoop-<PHD_SQOOP_VERSION>.noarch.rpm sqoop-metastore-<PHD_SQOOP_VERSION>.noarch.rpm
$ rpm -qa | grep sqoop
sqoop-<PHD_SQOOP_VERSION>.noarch
sqoop-metastore-<PHD_SQOOP_VERSION>.noarch
```

5. 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`.

You also need to restore sqoop environment files in `/etc/default` if they were changed.

6. Restart Sqoop Metastore service

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

Upgrade Oozie

1. Backup existing configuration files

```
$ cp -r /etc/gphd/oozie/conf <where to save Oozie configuration>
```

2. Stop Oozie Service

Before you run the upgrade, first stop Oozie service:

```
$ sudo service oozie stop
```

3. Upgrade Oozie Rpm's

```
$ sudo rpm -U oozie-client-<PHD_OOZIE_VERSION>.noarch.rpm oozie-<PHD_OOZIE_VERSION>.noarch.rpm
$ sudo rpm -qa | grep oozie
oozie-client-<PHD_OOZIE_VERSION>.noarch
oozie-<PHD_OOZIE_VERSION>.noarch
```

4. Clean up tomcat container files

```
$ sudo rm -rf /usr/lib/gphd/oozie-<PHD_OOZIE_OLD_VERSION>
```



PHD_OOZIE_OLD_VERSION is the old Oozie version in previous PHD release.

5. Copy Oozie Data

```
$ sudo -u oozie cp -r /var/lib/gphd/oozie-<PHD_OOZIE_OLD_VERSION>/* /var/lib/gphd/oozie/
```



PHD_OOZIE_OLD_VERSION is the old Oozie version in previous PHD release. The default OOZIE_DATA is "/var/lib/gphd/oozie/". If OOZIE_DATA is set to another path, you have to put it into the path you set.

6. Restore old configurations

After upgrading all Oozie packages, you need to restore your configuration files in /etc/gphd/oozie/conf

Check your previous backup configuration files, and manually add any change you made back to your upgraded configuration files.

You also need to restore oozie environment files in /etc/default if they were changed.

7. upgrade oozie database

```
$ sudo -u oozie oozie-setup db upgrade -run
$ sudo -u oozie oozie-setup db postupgrade -run
```

8. Init new version Oozie Service

```
$ sudo -u oozie oozie-setup prepare-war -d <ext-js-2.2-path>
$ sudo -u oozie hdfs dfs -rm -r share
Deleted share
$ sudo -u oozie oozie-setup sharelib create -fs hdfs://<NameNodeHost>:<HDFSPort>
```

Note: Make sure that there is a file named "ext-2.2.zip" in <ext-js-2.2-path >.

9. Restart Oozie service

```
$ sudo service oozie start
```

Upgrade HCatalog

1. Backup existing configuration files

```
$ cp -r /etc/gphd/hcatalog/conf <where to save HCatalog configuration>
$ cp -r /etc/default/hcatalog* <where to save HCatalog environment>
$ cp -r /etc/default/webhcat* <where to save Webhcat environment>
```

2. Stop webhcat-server Service

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

3. Stop hcatalog-server Service

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

4. Upgrade webhcat-server

```
$ rpm -U webhcat-server-<PHD_HCATALOG_VERSION>.noarch.rpm
$ rpm -qa | grep webhcat-server
webhcat-server-<PHD_HCATALOG_VERSION>.noarch.rpm
```

5. Upgrade webhcat

```
$ rpm -U webhcat-<PHD_HCATALOG_VERSION>.noarch.rpm
$ rpm -qa | grep webhcat
webhcat-<PHD_HCATALOG_VERSION>.noarch.rpm
```

6. Upgrade hcatalog-server

```
$ rpm -U hcatalog-server-<PHD_HCATALOG_VERSION>.noarch.rpm
$ rpm -qa | grep hcatalog-server
hcatalog-server-<PHD_HCATALOG_VERSION>.noarch.rpm
```

7. Upgrade hcatalog

```
$ rpm -U hcatalog-<PHD_HCATALOG_VERSION>.noarch.rpm
$ rpm -qa | grep hcatalog
hcatalog-<PHD_HCATALOG_VERSION>.noarch.rpm
```

8. Restore old configurations

After upgrading all packages, you need to restore your configuration files in `/etc/gphd/hcatalog/conf`.

Check your previous backup configuration files, and manually add any change you made back to your upgraded configuration files.

You also need to restore hcatalog and webhcat environment files in `/etc/default` if they were changed.

9. Restart webhcat-server service

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

10. Restart hcatalog-server service

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


Chapter 4 Pivotal Hadoop Enhancements

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

Topics:

- HDFS Off-Cluster Client Rack Awareness
 - Usage
- Vaidya
 - Overview
 - Installing Vaidya Files
 - Enabling Vaidya
 - Disabling Vaidya
 - Using Vaidya to Analyze Jobs
 - Vaidya Configuration Rules
 - Disabling a Rule
 - Changing the Importance of a Rule
 - Changing Success Threshold
 - Changing Input Parameters
 - Other
 - Adding a New Rule
 - Creating a Java Binary for a New Rule
 - Creating XML Configuration For a New Rule
 - Deploying files
- HVE Topology Awareness
 - Topology Awareness Configuration and Verification
- HVE Elasticity
 - Overview

- Function List
- Configuration
- Command Line Interface for YARN cluster
 - List all CLIs
 - List Nodes
 - Get Node Status
 - Set node resource capacity

HDFS Off-Cluster Client Rack Awareness

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:

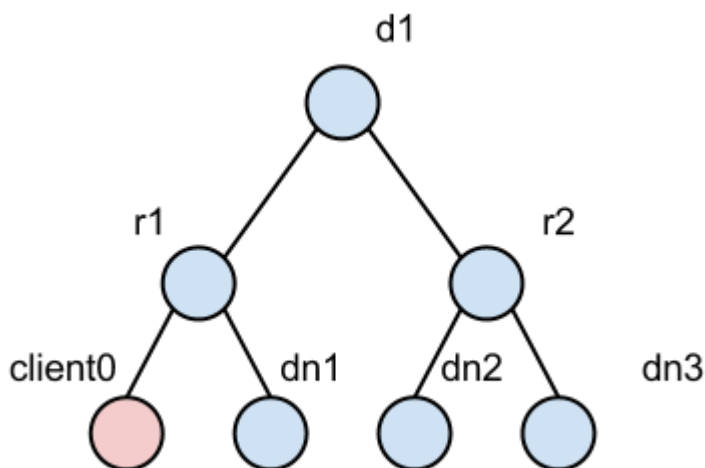
- BlockPlacementPolicy for write locality: namenode will look up network topology and construct a list of chosen nodes (pipeline) for requesting a block to locate, based on algorithms provided by a BlockPlacementPolicy.
- Block pseudo distance sort for read locality: when reading a block, after obtaining all the located blocks, namenode sorts these located blocks based on their topological distance from the 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 does not support off-cluster clients. The figure below is an example of off-cluster clients:



In this 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 large portion of I/O will be off-cluster client operations, which cannot benefit from localized I/O.
- When a physical cluster is not dedicated to Hadoop: a physical cluster might not be dedicated to Hadoop and other supporting systems, such as data loading tools, might 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 could be even more common in virtualized environments.

Solution: Design

To tackle this problem, we changed the logic in the block placement policy and the 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 the 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 that 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 that 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
client3	null	black list

The size of the 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 a random node that 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

Usage

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

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

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

Vaidya

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 the 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 reason for failures.

The Vaidya tool includes diagnostic rules (also referred to as "tests") where each rule analyzes a specific problem with the M/R job. A 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 uses the 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.

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 configuration file is installed into `/etc/gphd/hadoop/conf/`

Enabling Vaidya

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

Restart the job history server service to ensure the change takes effect.

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-voidya.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-mapreduce/hadoop-vaidya-<HADOOP_PHD_VERSION>.jar`.
- Restart the job history server service to ensure the change takes effect.

Using Vaidya to Analyze Jobs

1. Ensure your job history server service is running.
2. Successfully run a map-reduce job for Vaidya to analyze.
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 for the job.
 5. On the left side of the navigation area, there should be a link called **Vaidya report** under the navigation item **Job**. Click the **Vaidya report** link and Vaidya will analyze the job for you and show a report.

Vaidya Configuration Rules

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

You can find all rules to be run on a selected job in this 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 the 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 entire `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 the `Importance` element in the XML file. A level serves as a factor, which is multiplied to impact the 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 an 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 specified 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 straightforward 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 the threshold is compared with the impact value, rather than severity (which means making 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 might 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 instructions about the meaning of each XML element, as well as how to change them, refer to 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.vaidya.DiagnosticTest` class:

myrule.java

```
package com.greenplum.vaidya.rules;
import org.apache.hadoop.vaidya.DiagnosticTest;
import org.apache.hadoop.vaidya.statistics.job.JobStatistics;
public class MyRule extends DiagnosticTest {
    @Override
    public String getReferenceDetails() {
```

```

return "";
}
@Override
public String getPrescription() {
return "";
}
@Override
public double evaluate(JobStatistics jobStatistics) {
return 0.5;
}
}

```

3. Edit the three methods `getReferenceDetails`, `getPrescription` and `evaluate` to construct the logic. `evaluate` method should return a **double** value between 0.0 and 1.0 and represents the impact as the analysis result.
 - `getPrescription` method should return some text providing user suggestions/remedies about how to optimize your Map/Reduce configuration accordingly.
 - `getReferenceDetails` method should return some text indicating the meaningful counters and their values which can help you to diagnose your Map/Reduce configuration accordingly.
4. Compile the java class and package compiled class to a jar file, for example, `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.jar:/usr/lib/gphd/hadoop-mapreduce/lib/myrule
</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 `theFile.pathSeparator` attribute of your java platform to verify it.
- To make your settings take effect, restart the job history server service.

HVE 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 NodeManagers/TaskTrackers exist in separate virtual machines in virtualized environments, in order 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.

Topology Awareness Configuration and Verification

Sample Setup

This setup has 2 logical racks, 2 physical hosts (installed by ESXi and managed by vCenter) per rack, and 2 DN/NM (VM in ESXi) nodes per host. There is also one NameNode/ResourceManager and a client node that can be used to start jobs.

In this setup, each DN/NM node has 4 vCPUs, 16G memory, and 200G (Non-SSD) disks.

The NameNode and ResourceManager are installed on another dedicated VM with 4vCPU, 4G Memory and 100G disks.

Node Distribution on Hosts:

Rack 1	Host 1	NameNode and ResourceManager	DN1
	Host 2	DN2	DN3
Rack 2	Host 3	DN4	DN5
	Host 4	DN6	DN7

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>!-- point to topology.sh
location.-->
</property>
<property>
```

```

<name>net.topology.impl</name>
<value>org.apache.hadoop.net.NetworkTopologyWithNodeGroup</value>
<description> The default implementation of NetworkTopology which is classic three layer one.
</description>
</property>
<property>
  <name>net.topology.nodegroup.aware</name>
  <value>true</value>
  <description> By default, network topology is not aware of nodegroup layer.
  </description>
</property>
<property>
  <name>dfs.block.replicator.classname</name>
  <value>org.apache.hadoop.hdfs.server.blockmanagement.BlockPlacementPolicyWithNodeGroup</value>
  <description> The default implementation of ReplicationTargetChooser.
  </description>
</property>

```

2. Add the following line to yarn-site.xml:

```

<property>
  <description>The class to use as scheduled requests.</description>
  <name>yarn.resourcemanager.scheduled.requests.class</name>
  <value>org.apache.hadoop.mapreduce.v2.app.rm.ScheduledRequestsWithNodeGroup</value>
</property>
<property>
  <description> The boolean value to identify if the cluster is deployed on an environment which
needs an additional layer (node group) between node and rack for network topology.
  </description>
  <name>net.topology.with.nodegroup</name>
  <value>true</value>
</property>
<property>
  <description>The class to use as AbstractSchedulerElementsFactory in RM
scheduler.</description>
  <name>yarn.resourcemanager.scheduler.elements.factory.impl</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.SchedulerElementsFactoryWithNodeGro

```

Topology.data sample:

```

[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
```

HVE Elasticity

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

Overview

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

With this feature, APIs (CLI and JMX interface) and script tools are provided to get/set resources (memory, v-cores) on Hadoop cluster nodes for MR jobs.

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. The node status including its memory resource, v-core resource, hostname, health status, etc.
Set resource capacity in Node Manager node	Set resource capacity (memory, v-cores) of Node Manager to a node specified by user via CLI or JMX interface.

Configuration

To enable elastic resources, make the following changes to the Hadoop configuration.

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

```
<property>
  <name>yarn.dynamic.resource.enable</name>
  <value>true</value>
</property>
```

Command Line Interface for YARN cluster

List all CLIs

```
yarn rmdadmin
    [-refreshQueues]
    [-refreshNodes]
    [-refreshUserToGroupsMappings]
    [-refreshSuperUserGroupsConfiguration]
    [-refreshAdminAcls]
    [-refreshServiceAcl]
    [-getGroups [username]]
    [-updateNodeResource [NodeID][MemSize][Cores]]
    [-help [cmd]]

yarn node
  -all           Works with -list to list all nodes.
  -list          List all running nodes. Supports optional use of
                  -states to filter nodes based on node state, all -all
                  to list all nodes.
  -states <States> Works with -list to filter nodes based on input
                  comma-separated list of node states.
  -status <NodeId> Prints the status report of the node.
```

List Nodes

```
yarn node -list -all
```

Get Node Status

```
yarn node -status <NodeId>
```

Set node resource capacity

```
yarn rmdadmin -updateNodeResource [NodeID][MemSize][Cores]
```

Example

List all nodes

```
yarn node -list -all
```



```
Total Nodes:8
      Node-Id      Node-State  Node-Http-Address  Number-of-Running-Containers
hdsh2-a172.lss.emc.com:37804      RUNNING  hdsh2-a172.lss.emc.com:8042
0
hdsh2-a173.lss.emc.com:45310      RUNNING  hdsh2-a173.lss.emc.com:8042
0
hdsh2-a159.lss.emc.com:60596      RUNNING  hdsh2-a159.lss.emc.com:8042
0
hdsh2-a158.lss.emc.com:51694      RUNNING  hdsh2-a158.lss.emc.com:8042
0
hdsh2-a157.lss.emc.com:37348      RUNNING  hdsh2-a157.lss.emc.com:8042
0
hdsh2-a174.lss.emc.com:33263      RUNNING  hdsh2-a174.lss.emc.com:8042
0
hdsh2-a160.lss.emc.com:34460      RUNNING  hdsh2-a160.lss.emc.com:8042
0
hdsh2-a171.lss.emc.com:49469      RUNNING  hdsh2-a171.lss.emc.com:8042
0
```

Get node status

```
yarn node -status hdsh2-a172.lss.emc.com:37804
```

```
Node Report :
```

```
Node-Id : hdsh2-a172.lss.emc.com:37804
Rack : /dc/rc2
Node-State : RUNNING
Node-Http-Address : hdsh2-a172.lss.emc.com:8042
Last-Health-Update : Wed 15/Jan/14 03:38:08:402CST
Health-Report :
Containers : 0
Memory-Used : 0MB
Memory-Capacity : 2048MB
CPU-Used : 0 vcores
CPU-Capacity : 2 vcores
```

Set node resource capacity

```
yarn rmadmin -updateNodeResource hdsh2-a172.lss.emc.com:37804 2048 2
```

Chapter 5 Security

You must install and configure Kerberos to enable security in Pivotal HD 1.1.x. and higher.

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

Topics:

- Configuring Kerberos for HDFS and YARN (MapReduce)
 - Kerberos Set-up
 - Java Support Items Installation
 - Container and Script Modifications
 - Site XML Changes
 - Complete the HDFS/YARN Secure Configuration
 - Turning Secure Mode Off
 - Building and Installing JSVC
 - Installing the MIT Kerberos 5 KDC
- Configuring Kerberos for HDFS High Availability
- Zookeeper Secure Configuration
 - Zookeeper Servers
 - Zookeeper Clients
- HBase Secure Configuration
 - HBase Master and RegionServers
 - HBase Clients
 - HBase with Secure Zookeeper Configuration
 - Access Control and PXF External Tables
- Hive Secure Configuration
 - Changing to Hive Server 2
 - Hive warehouse permissions issues

- Connecting and using secure Hive with Beeline
- Configure HCatalog (WebHCat) on secure Hive
 - Prerequisites
 - Create keytab file for the WebHCat server
 - Distribute the keytab file to the WebHCat server
 - Configure WebHCat and proxy users
 - Verify WebHCat is working
- HAWQ on Secure HDFS
 - Requirements
 - Preparation
 - Configuration
 - Troubleshooting
- Auditing
- Secure Web Access
 - Overview
 - Prerequisites
 - Configuring Secure WebHDFS
 - Using WebHDFS in Secure Mode
- Secure HDFS web access via HttpFS
 - Prerequisites
 - Add principal for HttpFS
 - Create and distribute keytab
 - Set the keytab file ownership and permissions
 - Configuration
 - Restart HttpFS
 - Verify it's working
- Flume Security Configuration

- Prerequisites
- Create the Flume Principal
- Create the Flume Keytab Files
- Distribute the Flume Keytab Files to the Flume server and change the ownership and permission
- A single user for all HDFS sinks
- Different users across multiple HDFS sinks
- Oozie Security Configuration
 - Prerequisites
 - Create the Oozie Principal
 - Create the HTTP Principal for the Oozie Server
 - Create the Oozie Keytab Files
 - Copy the Oozie Keytab Files to the Oozie server and change the ownership and permission
 - Edit the Oozie Configuration
 - Using Oozie with a Secure Hive Metastore Server
 - Verify Secure Oozie
- Sqoop Security Configuration
- Pig Security Configuration
- Mahout Security Configuration
- Troubleshooting

**Notes**

- For HAWQ to work with secure HDFS, the Pivotal ADS version must be 1.1.3 or greater.
- For more information about HAWQ secure configuration, see the *Kerberos Authentication* section of the *Pivotal ADS Administrator Guide*.
- Note that Kerberos operation in Hadoop is very sensitive to proper networking configuration:
 - Host IP's for service nodes must reverse map to the FQDN's used to create the node principal for the service/FQDN.
 - `hostname -f` on a node must give the FQDN used to create the principal for the service/FQDN.

- The cluster needs to have been created with FQDN's, not short names.

Make sure your networking is properly configured before attempting to secure a cluster.

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.

Kerberos Set-up

Install the KDC

If you do not have a pre-existing KDC, see [Installing the MIT Kerberos 5 KDC](#).

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

Install Kerberos Workstation and Libraries on Cluster Hosts

If you are using MIT krb5 run:

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

Distribute the Kerberos Client Configuration File to all Cluster Hosts

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

Create the Principals

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

Principals (Kerberos users) are of the form: `name/role@REALM`. For our purposes the name will be a PHD service name (for example, `hdfs`), and the role will be a DNS resolvable 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, proxy server).

For MapReduce services, you need to create a principal, `mapred/host_fqdn` for the Job History Server.

To create the required secure HD principals (running `kadmin.local`):

- For each cluster host (excepting client-only hosts) run:

```
addprinc -randkey HTTP/<host_fqdn>@<REALM>
```

- HDFS (name node, secondary name node, data nodes), for each HDFS service host run:

```
addprinc -randkey hdfs/<host_fqdn>@<REALM>
```

- YARN (resource manager, node managers, proxy server), 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 that you use a cluster-local KDC for this purpose and configure cross-realm trust to your organizational Active Directory or other Kerberos KDC.

Create the Keytab Files

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

Important: You can put the keytab files anywhere during this step. In this document, we created a directory `/etc/security/phd/keytab/` and are using this directory 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/phd/keytab/`, it may be advisable 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 (name node, secondary name node, data nodes), run:

```
kadmin.local: ktadd -norandkey -k /etc/security/phd/keytab/hdfs-hostid.service.keytab
hdfs/<host_fqdn>@<REALM> HTTP/<host_fqdn>@<REALM>
```

where `hostid` is the 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/phd/keytab/hdfs-vm2.service.keytab
hdfs/centos62-2.localdomain@BIGDATA.COM HTTP/centos62-2.localdomain@BIGDATA.COM
kadmin.local: ktadd -norandkey -k /etc/security/phd/keytab/hdfs-vm3.service.keytab
hdfs/centos62-3.localdomain@BIGDATA.COM HTTP/centos62-3.localdomain@BIGDATA.COM
kadmin.local: ktadd -norandkey -k /etc/security/phd/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, node manager or proxy server), run:

```
kadmin.local: ktadd -norandkey -k /etc/security/phd/keytab/yarn-hostid.service.keytab
yarn/<host_fqdn>@<REALM> HTTP/<host_fqdn>@<REALM>
```

For example, for a three node cluster (one node resource manager, two node managers):

```
kadmin.local: ktadd -norandkey -k /etc/security/phd/keytab/yarn-vm2.service.keytab
yarn/centos62-2.localdomain@BIGDATA.COM HTTP/centos62-2.localdomain@BIGDATA.COM
kadmin.local: ktadd -norandkey -k /etc/security/phd/keytab/yarn-vm3.service.keytab
yarn/centos62-3.localdomain@BIGDATA.COM HTTP/centos62-3.localdomain@BIGDATA.COM
kadmin.local: ktadd -norandkey -k /etc/security/phd/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/phd/keytab/mapred-hostid.service.keytab
mapred/host_fqdn@REALM HTTP/host_fqdn@REALM
```

For example:

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

Distribute the Keytab Files

1. On each cluster node, create the directory for the keytab files; here, we are using `/etc/security/phd/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. a. Change the permissions on all key tabs to read-write by owner only:


```
chmod 400 *.keytab
```
 - b. Change the group on all keytab files to `hadoop`:


```
chgrp hadoop *
```
 - c. Change the owner of each keytab to the relevant principal name.
For example, for `yarn-vm2.service.keytab` run: `chown yarn yarn-vm2.service.keytab`

- d. Create links to the files of the form `principalname.service.keytab` .

For example, for `yarn-vm2.service.keytab` run:

```
ln -s yarn-vm2.service.keytab yarn.service.keytab
```



important

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

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, proxy server):

```
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
```

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).
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 actions:

a. Install JSVC using `yum` and run the check again; if it fails try the next step.

b. 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 perform step 2, above, another time. JSVC is very picky about platform and JDK matching, so use the [Building and Installing JSVC](#) instructions for your system OS and JDK.

Container and Script Modifications

Configure the Linux Container

1. Edit the `/usr/lib/gphd/hadoop-yarn/etc/hadoop/container-executor.cfg` 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 distribution; 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
```

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 perform 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`, it 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. 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.



As long as `HADOOP_SECURE_DN_USER` is, set the datanode will try to start in secure mode.

Site XML Changes

Using `_HOST` in Site XML

You can 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 login to Kerberos without a password, using a keytab file.

- Each service needs to know its own principal name.
- 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. 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 for PHD VERSION LOWER THAN 2.0 so
disable ONLY for them -->
<!-- For PHD greater than or equal to 2.0, set this to true -->

<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/phd/keytab/hdfs.service.keytab</value>
</property>

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

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

```

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

<!-- (optional) secondary name node secure configuration info -->

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

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

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

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

```

```

<property>
  <name>dfs.datanode.kerberos.http.principal</name>
  <value>HTTP/_HOST@REALM</value>
</property>

<property>
  <name>dfs.datanode.keytab.file</name>
  <value>/etc/security/phd/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/_HOST@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/phd/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/_HOST@REALM</value>
</property>

```



```

<property>
  <name>yarn.resourcemanager.keytab</name>
  <value>/etc/security/phd/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/_HOST@REALM</value>
</property>

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

<!-- OPTIONAL - set these to enable secure proxy server node -->

<property>
  <name>yarn.web-proxy.keytab</name>
  <value>/etc/security/phd/keytab/yarn.service.keytab</value>
</property>

<property>
  <name>yarn.web-proxy.principal</name>
  <value>yarn/_HOST@REALM</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/phd/keytab/mapred.service.keytab</value>
</property>

<property>
  <name>mapreduce.jobhistory.principal</name>
  <value>mapred/_HOST@REALM</value>
</property>

```

Complete the HDFS/YARN Secure Configuration

1. Start the cluster:

```
$ icm_client start
```

2. Check that all the processes listed below 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 (the 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`):
 - a. Login as `gpadmin` on any HDFS service node (namenode, datanode).
 - b. Execute the following:

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

- c. Execute the following:

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

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

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

If this all works, then you are ready to configure other services. If not, see the [Troubleshooting](#) section.

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:

- a. If you made backups as suggested above:

Restore the original site xml files

or:

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

- • • Set the Linux container executable to `org.apache.hadoop.yarn.server.nodemanager.DefaultContainerExecutor` 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.
- Undo the changes to the Zookeeper site xml and configuration files.
- If applicable, revert the changes to the `hdfs-client.xml` and `gpinisystem_config` for HAWQ.
- If applicable, undo the changes to the Hive and HBase site xml, configuration, and environments.
- Start the cluster.

Building and Installing JSVC

In order for the data nodes to start as root to get secure ports, then switch back to the hdfs user, jsvc must be installed (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 perform the make on one node, then the binary can be distributed to the others (assuming all systems are the same basic image):

1. Install gcc and make (you can remove them after this process if desired):

```
yum install gcc make
```

2. Download the Apache commons daemon. For example, commons-daemon-1.0.15-src.zip was tested. The demon is available here: http://commons.apache.org/proper/commons-daemon/download_daemon.cgi
3. scp it to one of your 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 was 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`.

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`
- Set the maximum ticket lifetime and renew lifetime to your desired values (24 hours and 7 days are typical)
- 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
    max_life = 24h 0m 0s
    max_renewable_life = 7d 0h 0m 0s
    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:

- a. Login as the admin you created: `kinit username/admin`

- b. run `kadmin` and make sure you can login

If you get the message `kinit: Cannot contact any KDC for realm 'REALM' while getting initial credentials`, then the KDC is not running or the KDC host information in `/etc/kdc.conf` is incorrect.

You should now have a KDC that is functional for PHD secure cluster operations.

Configuring Kerberos for HDFS High Availability



Currently we only support Quorum Journal-based storage for high availability.

To configure Kerberos for HDFS HA, add the following Quorum Journal-based storage configuration properties to the `hdfs-site.xml` file on all machines in the cluster:

```
<property>
  <name>dfs.journalnode.keytab.file</name>
  <value>/etc/security/phd/keytab/hdfs.keytab</value> <!-- path to the HDFS keytab -->
</property>
<property>
  <name>dfs.journalnode.kerberos.principal</name>
  <value>hdfs/_HOST@REALM.COM</value>
</property>
<property>
  <name>dfs.journalnode.kerberos.internal.spnego.principal</name>
  <value>HTTP/_HOST@REALM.COM</value>
</property>
```


Zookeeper Secure Configuration

Zookeeper secure configuration for server is recommended for HBase.

Important: STOP cluster services before doing this configuration.

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/phd/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/phd/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/phd/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"
```

If JVMFLAGS already exist, then modify that and add new value within quotes separated by space. For example, modify `"export JVMFLAGS="-Xmx2048m"` to:

```
export JVMFLAGS="-Xmx2048m -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
```

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/phd/keytab/zclient-hostid.client.keytab
zclient/host_fqdn@REALM
```

3. Move the file to the `/etc/security/phd/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/phd/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/phd/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.

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>).

HBase Master and Regionserver

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/phd/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/phd/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/_HOST@REALM</value>
  </property>

  <property>
    <name>hbase.regionserver.keytab.file</name>
    <value>/etc/security/phd/keytab/hbase.service.keytab</value>
  </property>

  <!-- HBase secure master configuration -->
  <property>
    <name>hbase.master.kerberos.principal</name>
    <value>hbase/_HOST@REALM</value>
  </property>

  <property>
    <name>hbase.master.keytab.file</name>
    <value>/etc/security/phd/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 Regionserver 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.

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.

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.

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/phd/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
```

Access Control and PXF External Tables

The version of HBase distributed with PHD supports access control. Basic mappings of permissions (RWCA) to operations allowed:

permission	operations
Read	
	Get
	Exists
	Scan
Write	
	Put
	Delete
	Lock/UnlockRow
	IncrementColumnValue
	CheckAndDelete/Put
	Flush
	Compact
Create	
	Admin
	Alter
	Drop
Admin	
	Enable/Disable
	>Snapshot/Restore/Clone
	Split
	Major Compact
	Grant
	Revoke
	Shutdown

Permission scopes:

Table:

- Read: User can read from any column family in table
- Write: User can write to any column family in table
- Create: User can alter table attributes; add, alter, or drop column families; and drop the table.
- Admin: User can alter table attributes; add, alter, or drop column families; and enable, disable, or drop the table. User can also trigger region (re)assignments or relocation.

Column Family:

- Read: User can read from the column family
- Write: User can write to the column family

For PHD the Hbase superuser is the "hbase" user/principal. There is an implicit global scope for permissions granted by the superuser.

Tables have an OWNER attribute that defaults to the table creator but may be changed with an Alter operation.

See the HBase documentation here: <http://hbase.apache.org/book/hbase.accesscontrol.configuration.html> for further details on configuring and using access controls.

Set the properties show below to enable access control:

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

<property>
  <name>hbase.coprocessor.master.classes</name>
  <value>org.apache.hadoop.hbase.security.access.AccessController</value>
</property>

<property>
  <name>hbase.coprocessor.region.classes</name>
  <value>org.apache.hadoop.hbase.security.access.AccessController,
org.apache.hadoop.hbase.security.token.TokenProvider</value>
</property>
```

When access control is enabled you will need to explicitly grant users permissions to the database as shown below (example assumes it is being run on an HBase service host). You may also choose to create a specific hbase principal (hbase@REALM) for HBase administration rather than using the service keytab as shown below; this would allow administration from any client host.

```
# sudo -u hbase kinit -kt /etc/security/phd/keytab/hbase.service.keytab hbase/<host_fqdn>
# sudo -u hbase hbase shell
hbase> grant 'myuser', 'RWCA'
# kdestroy
```

PXF Access

If you are using PXF external HBase tables you will need to grant user 'hdfs' permissions as above.

```
# sudo -u hbase kinit -kt /etc/security/phd/keytab/hbase.service.keytab hbase/<host_fqdn>
# sudo -u hbase hbase shell
hbase> grant 'hdfs', 'RWCA'
# kdestroy
```



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-site.xml` configuration file to true, as shown in the example below.



Hive Server version 1 (hiveserver) does not support Kerberos, so you will need to switch to using Hive Server 2 in a secured cluster. See the instructions below.

1. Add the Kerberos principals and their locations to the `hive-site.xml`. For example:

```
<property>
  <name>hive.server2.authentication</name>
  <value>KERBEROS</value>
</property>

<property>
  <name>hive.server2.authentication.kerberos.principal</name>
  <value>hive/_HOST@REALM</value>
</property>

<property>
  <name>hive.server2.authentication.kerberos.keytab</name>
  <value>/etc/security/phd/keytab/hive.keytab</value>
</property>

<property>
  <name>hive.server2.enable.impersonation</name>
  <value>true</value>
</property>

<property>
  <name>hive.server2.enable.doAs</name>
  <value>true</value>
</property>

<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.security.authorization.enabled</name>
  <value>true</value>
  <description>enable or disable the hive client authorization</description>
</property>
```

```

<property>
  <name>hive.security.authorization.createtable.owner.grants</name>
  <value>ALL</value>
  <description>the privileges automatically granted to the owner whenever a table gets created.
    An example like "select,drop" will grant select and drop privilege to the owner of the table.
    You may change this value if you desire lower privileges on create.</description>
</property>

<property>
  <name>hive.metastore.kerberos.keytab.file</name>
  <value>/etc/security/phd/keytab/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@REALM</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>

```

2. Add the following parameters to `core-site.xml` (Hadoop configuration) to enable users to run hive queries:

```

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

<!-- you may of course add more restrictive values here -->

```

Changing to Hive Server 2

You will need to change to Hive Server 2 in order to use Hive in non-local mode with Kerberos. On the Hive server edit the file `/etc/gphd/init.d/hive-server` and change the line:

```
server_name="server"
```

To:

```
server_name="server2"
```

Then restart the Hive Server:

```
/etc/init.d/hive-server restart.
```

Hive warehouse permissions issues

As of this version there is a known issue with Hive permissions in that the default 775 warehouse permissions on `/hive/gphd/warehouse` will prevent access by non-superuser users for writing. The simplest way to resolve this is, as the `hdfs` user, change the permissions to 777. If this is not acceptable then Hive external tables may provide a solution.



You will need to have created an `hdfs` principal to do HDFS administration work such as changing permissions on other users' or root level directories.

Connecting and using secure Hive with Beeline

In order to use secure Hive you should create a Kerberos principal for the `hive` user; this is your Hive administrator principal.

Whenever you want to grant privileges to an ordinary user you need to be logged in as the `hive` superuser.

To log in as a particular user using beeline:

1. Make sure there are no tickets for any other user in your cache (running `kdestroy` and the `kinit` user will insure this).
2. `kinit` as the Hive user you want to log in as.
3. Connect to beeline using the form:

```
!connect
```

```
jdbc:hive2://<hive_server2_fqdn>:10000/<database>;principal=hive/<hive_server2_fqdn>
```

where:

`<hive_server2_fqdn>` is the FQDN of the host for the Hive Server role.

`<database>` is the database to connect to (for example, default).

`REALM` is your Kerberos realm.



- Unless you already have done `kinit` as the user you want to connect as, and have insured only that user has a ticket in the cache (using `klist`), you will need to enter your kerberos principal name and password twice: once before starting beeline by doing `kinit`, and again when queried by Hive during connect.

- Remember that in order for an ordinary user to do anything you must first follow the above sequence for user "hive" (Hive administrator) and GRANT the user privileges.
- As of the version tested SHOW GRANT does not show any actual information in beeline, so you can only tell if a GRANT worked by testing as that user (or looking for errors in the hive log).
- GRANTing as another user will not have any affect on databases not owned by that user, so if your grant create permissions in database default as user "tester", for example, that user will not actually get permissions to database "default" (and you may get no error- so be aware).

Configure HCatalog (WebHCat) on secure Hive

HCatalog is a tool that operates on the Hive metastore. If you want to use the HCatalog RESTful APIs, aka., WebHCat, security configuration is required to enable the security functionality of the WebHCat server.

Prerequisites

- Hive is installed and configured properly on your cluster.
- Hive metastore is configured to work in remote mode.
- Security is enabled for Hive properly, as shown in the [Hive Secure Configuration](#) section.
- HCatalog and WebHCat are installed and configured properly on your cluster.

Create keytab file for the WebHCat server

Create a keytab file that contains the HTTP principal:

```
> kadmin.local
kadmin.local: ktadd -norandkey -k /etc/security/phd/keytab/webhcat-hostid.service.keytab
HTTP/host_fqdn@REALM
```

Distribute the keytab file to the WebHCat server

Copy the generated `/etc/security/phd/keytab/webhcat-hostid.service.keytab` file to the machine where the WebHCat server is running, put it under `/etc/security/phd/keytab/`. Then create symbolic link and adjust file owner and permissions:

```
> chown hive:hadoop /etc/security/phd/keytab/webhcat-hostid.service.keytab
> chmod 400 /etc/security/phd/keytab/webhcat-hostid.service.keytab
> ln -s /etc/security/phd/keytab/webhcat-hostid.service.keytab
/etc/security/phd/keytab/webhcat.service.keytab
```

Configure WebHCat and proxy users

On the WebHCat server machine, edit the file `/etc/gphd/webhcat/conf/webhcat-site.xml` and add the following properties:

```
webhcat-site.xml

<property>
  <name>templeton.kerberos.secret</name>
  <value>SuPerS3c3tV@lue!</value>
</property>
```



```
<property>
  <name>templeton.kerberos.keytab</name>
  <value>/etc/security/phd/keytab/webhcat.service.keytab</value>
</property>

<property>
  <name>templeton.kerberos.principal</name>
  <value>HTTP/_HOST@<REALM></value>
</property>
```

Note in the above snippet that you need to replace `<FQDN>` with the correct FQDN of your WebHCat server machine, replace `<REALM>` with the Kerberos realm name you are using for your cluster. You may set the `templeton.kerberos.secret` property value to any random secret value.



In most cases when the WebHCat server starts it reads the Hive metastore information from the corresponding Hive configuration files (`/etc/gphd/hive/conf/hive-site.xml`). If your WebHCat server is not running on the same machine as Hive server, you may need manually copy `/etc/gphd/hive/conf/hive-site.xml` from the Hive server machine to WebHCat server machine before restarting the WebHCat server.

After edit the `webhcat-site.xml` file, you need restart WebHCat server to make the changes take effect.

```
> service webhcat-server restart
```

Because WebHCat will access the HDFS NameNode, you need configure Hadoop to allow impersonation as the HTTP user. Edit `/etc/gphd/hadoop/conf/core-site.xml` on all your HDFS node machines, add the following content if they are not already there:

```
core-site.xml

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

After you edited `core-site.xml`, restart all HDFS services to make the change work.

Verify WebHCat is working

Ensure you have a version of `curl` installed that supports GSS-negotiation by calling `curl -V`:

```
> curl 7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19.7 NSS/3.13.6.0 zlib/1.2.3 libidn/1.18  
libssh2/1.4.2  
Protocols: tftp ftp telnet dict ldap ldaps http file https ftps scp sftp  
Features: GSS-Negotiate IDN IPv6 Largefile NTLM SSL libz
```

Then do:

```
> kinit <username>  
<output omitted, you will need enter password to login>  
> curl -i -u : --negotiate 'http://<FQDN>:<PORT>/templeton/v1/ddl/database'  
HTTP/1.1 401  
WWW-Authenticate: Negotiate  
Set-Cookie: hadoop.auth=;Path=/;Expires=Thu, 01-Jan-1970 00:00:00 GMT  
Cache-Control: must-revalidate,no-cache,no-store  
Content-Type: text/html;charset=ISO-8859-1  
Content-Length: 1274  
Server: Jetty(7.6.0.v20120127)  
  
HTTP/1.1 200 OK  
Set-Cookie:  
hadoop.auth="u=root&p=root@EXAMPLE.COM&t=kerberos&e=1393939264464&s=mJG9x4mE4S9BMDCbSeRcVKyRY5g=";Ver  
Thu, 01 Jan 1970 00:00:00 GMT  
Content-Type: application/json  
Transfer-Encoding: chunked  
Server: Jetty(7.6.0.v20120127)  
  
{ "databases": [ "default" ] }
```

In the above snippet, replace <FQDN> with the real FQDN of your WebHCat server machine, and replace <PORT> with your WebHCat server port (by default, it's 50111).

If you see output similar to the above (a 401 response followed by a 200 one), your secured WebHCat installation is functioning properly.

HAWQ on Secure HDFS

Requirements


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

Preparation

1. If HAWQ is already initialized and running, stop HAWQ by running `service hawq stop` or `<HAWQ installation directory>/bin/gpstop`.
2. Secure the HDFS cluster using the instructions provided in the *Pivotal HD Stack and Tool Reference Guide* or using available security tools.
3. Insure HDFS is running properly in secured mode.
4. Insure that the property `dfs.encrypt.data.transfer` is set to `false` in the `hdfs-site.xml` for your cluster.

Configuration

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

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

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

2. Move this keytab file to the appropriate keytab directory on the HAWQ master node (for example, `/etc/security/phd/keytab/`).
3. Set the ownership of the keytab file to `gpadmin:gpadmin` and the permissions to `400`.

4. Refer to your `gpinitssystem_config` file (typically in `/etc/gphd/hawq/conf`) to determine your configured HAWQ HDFS data directory (typically `/hawq_data`). This will be the last part of the `DFS_URL` value. For example if `DFS_URL` is set to `centos61-2:8020/hawq_data` then your HAWQ HDFS data directory is `/hawq_data`.
5. Create (if required) the HAWQ HDFS data directory in HDFS, and assign ownership as `postgres:gpadmin` and permissions `755`.



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

6. Create in HDFS (if not present) the directory `/user/gpadmin` with ownership `gpadmin:gpadmin` and permissions `777`.
7. Modify the `hdfs-client.xml` file (typically in `/usr/lib/gphd/hawq/etc`) on the master node and ALL segment server nodes by adding the following:

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

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



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

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



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

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

a. `service hawq start`

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

c. `gpconfig --masteronly -c krb_server_keyfile -v "'/path/to/keytab/file'"`

NOTE The single quotes `'` after/before the double quotes `"` in the keytab string above are required!

d. `service hawq stop`

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

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

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

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

Troubleshooting

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

- Incorrect values in your `hdfs-client.xml`
- `hdfs-client.xml` not updated on master and all segment servers
- Unable to login with Kerberos; possible bad keytab or principal for "postgres"
 - Validate on master by doing: `kinit -k <keytab dir path>/hawq.service.keytab postgres@EXAMPLE.COM`
- Wrong HAWQ HDFS data directory or directory permissions: Check your `gpinitssystem_config` file and the `DFS_URL` value and the directory permissions.

- Unable to create the HAWQ HDFS data directory errors: insure that you have created the proper directory as specified in `gpinitssystem_config` and that the ownership and permissions are correct.

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 2.0 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/`

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.

Secure Web Access

This section describes how to configure WebHDFS on a secure PHD cluster.

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"
      },
      ...
    ]
  }
}
```

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.

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
```

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. Do this by running `kinit` command with your user name, in this example, `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 Kerberos'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/EuqluQ9C2F6Yg8
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,"spaceConsumed"
```

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.

Secure HDFS web access via HttpFS

HttpFS is another set of RESTful APIs which enable you to operate HDFS via the HTTP protocol. It has API's compatible with WebHDFS, and you also need configuration changes to make it work with security.

Prerequisites

- Have Hadoop and HDFS installed and security enabled
- Have HttpFS installed correctly

Add principal for HttpFS

HttpFS needs two principals, one for secured HTTP access and another for secured Hadoop access. Since in secured Hadoop there is already an HTTP/<host FQDN>@<realm> principal for all HTTP access, only one new principal needs to be added. Run the following commands on KDC host (or use kadmin from another host):

```
> kadmin.local
kadmin.local : addprinc -randkey httpfs/host_fqdn@REALM
```

Create and distribute keytab

Put the new httpfs/host_fqdn@REALM principal and the existing HTTP/host_fqdn@REALM principal keys into one keytab file:

```
kadmin.local : ktadd -norandkey -k /etc/security/phd/keytab/httpfs-hostid.service.keytab
httpfs/host_fqdn@REALM HTTP/host_fqdn@REALM
```

The above command should create the file

/etc/security/phd/keytab/httpfs-hostid.service.keytab. Distribute this keytab file to the node where you installed HttpFS as /etc/security/phd/keytab/httpfs.<host_fqdn>.service.keytab.

Set the keytab file ownership and permissions

Owner and permissions need to be properly setup for the distributed keytab file to make it readable by HttpFS service. It is also recommended to make a symbol link as below:

```
> chown httpfs:hadoop /etc/security/phd/keytab/httpfs.<host_fqdn>.service.keytab
> chmod 400 /etc/security/phd/keytab/httpfs.<host_fqdn>.service.keytab
> ln -s /etc/security/phd/keytab/httpfs.<host_fqdn>.service.keytab
/etc/security/phd/keytab/httpfs.service.keytab
```

Configuration

Edit the file `/etc/gphd/hadoop-httpfs/conf/httpfs-site.xml` and add the following properties to the configuration file:

```
httpfs-site.xml

<property>
  <name>httpfs.authentication.type</name>
  <value>kerberos</value>
</property>
<property>
  <name>httpfs.hadoop.authentication.type</name>
  <value>kerberos</value>
</property>
<property>
  <name>httpfs.authentication.kerberos.principal</name>
  <value>HTTP/<host_fqdn>@<REALM></value>
</property>
<property>
  <name>httpfs.authentication.kerberos.keytab</name>
  <value>/etc/security/phd/keytab/httpfs.service.keytab</value>
</property>
<property>
  <name>httpfs.hadoop.authentication.kerberos.principal</name>
  <value>httpfs/<host_fqdn>@<REALM></value>
</property>
<property>
  <name>httpfs.hadoop.authentication.kerberos.keytab</name>
  <value>/etc/security/phd/keytab/httpfs.service.keytab</value>
</property>
<property>
  <name>httpfs.authentication.kerberos.name.rules</name>
  <value>DEFAULT</value>
  <!-- same as the value of hadoop.security.auth_to_local in your core-site.xml -->
</property>
```

Note that you should replace `<host_fqdn>@<REALM>` with your real FQDN and REALM in the above example and the property value of `httpfs.authentication.kerberos.name.rules` should be the same as the value of the property `hadoop.security.auth_to_local` in your `/etc/gphd/hadoop/conf/core-site.xml`.

Restart HttpFS

Next restart HttpFS service to apply the configuration changes:

```
> service hadoop-httpfs restart
```

Verify it's working

You need curl with GSS negotiation enabled to verify that secured HttpFS is working. Check your curl features with:

```
> curl -V
curl 7.19.7 (x86_64-unknown-linux-gnu) libcurl/7.19.7 NSS/3.12.7.0 zlib/1.2.3 libidn/1.18
libssh2/1.2.2
Protocols: tftp ftp telnet dict ldap ldaps http file https ftps scp sftp
Features: GSS-Negotiate IDN IPv6 Largefile NTLM SSL libz
```

Now you need to login as a user who has corresponding principal in the KDC, we use user `hadoop` as an example

```
> su - hadoop
hadoop> kinit
<output omitted, you need password to login>
hadoop> curl -i --negotiate -u : "http://<host_fqdn>:<PORT>/webhdfs/v1/user?op=LISTSTATUS"
HTTP/1.1 401 Unauthorized
Server: Apache-Coyote/1.1
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=""; Expires=Thu, 01-Jan-1970 00:00:10 GMT; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 951
Date: Wed, 26 Feb 2014 09:37:51 GMT

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie:
hadoop.auth="u=hadoop&p=hadoop@EXAMPLE.COM&t=kerberos-dt&e=1393443472388&s=18UnRj4g0DqUTHyPqC7kC3amsr
Version=1; Path=/
Content-Type: application/json
Transfer-Encoding: chunked
Date: Wed, 26 Feb 2014 09:37:51 GMT

{"FileStatuses":{"FileStatus":[{"pathSuffix":"hadoop","type":"DIRECTORY","length":0,"owner":"hadoop",
```

In the above example, please ensure you replace `<host_fqdn>` with real FQDN in your environment (it MUST be a FQDN, any short name or alias like `"localhost"` will not work), and `<PORT>` with the actual port (by default, it is 14000).

If you see output similar to the above (one 401 response and one 200 response followed), your secured HttpFS is working.

Flume Security Configuration

This section describes the Flume security configurations.

Prerequisites

- Flume must be installed on the cluster.
- Security has been enabled for HDFS on the cluster.

Create the Flume Principal

On the KDC admin server, create a principal for the flume server:

```
kadmin.local: addprinc -randkey flume/host_fqdn@REALM
```

Create the Flume Keytab Files

On the KDC admin server, create the Flume keytab files:

```
kadmin.local: ktadd -norandkey -k /etc/security/phd/keytab/flume-hostid.service.keytab  
flume/host_fqdn@REALM
```

Distribute the Flume Keytab Files to the Flume server and change the ownership and permission

Move the keytab file created in the previous step to the `/etc/security/phd/keytab` directory on the host running the Flume server, then run the following commands on the Flume server:

```
cd /etc/security/phd/keytab  
  
chgrp flume flume-hostid.service.keytab  
  
chown flume flume-hostid.service.keytab  
  
chmod 400 flume-hostid.service.keytab  
  
ln -s flume-hostid.service.keytab flume.service.keytab
```

A single user for all HDFS sinks

Add following properties in `/etc/gphd/flume/conf/flume.conf` in the Flume server:


```
agentName.sinks.sinkName.hdfs.kerberosPrincipal = flume-hostid/host_fqdn@REALM
agentName.sinks.sinkName.hdfs.kerberosKeytab = /etc/security/phd/keytab/flume.service.keytab
```

Flume configuration example:

```
agent.sources = r1
agent.sinks = k1
agent.channels = c1

# Describe/configure the source
agent.sources.r1.type = netcat
agent.sources.r1.bind = localhost
agent.sources.r1.port = 44444

# Describe the sink
agent.sinks.k1.type = hdfs
agent.sinks.k1.hdfs.path = hdfs://centos64-1.localdomain/user/flume
agent.sinks.k1.hdfs.fileType = DataStream
agent.sinks.k1.hdfs.kerberosPrincipal = flume/_HOST@REALM.COM
agent.sinks.k1.hdfs.kerberosKeytab = /etc/security/phd/keytab/flume.service.keytab

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

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

Different users across multiple HDFS sinks

The same keytab path must be used across all HDFS sinks in the same agent since Flume does not support using multiple Kerberos principals or keytabs in the same agent.

If multiple users on HDFS are used, impersonation in core-site.xml in Hadoop must be configured:

Configure impersonation in core-site.xml

```
<property>
  <name>hadoop.proxyuser.flume.groups</name>
  <value>group1,group2</value>
  <description>Allow the flume user to impersonate any members of group1 and
group2</description>
</property>
<property>
  <name>hadoop.proxyuser.flume.hosts</name>
  <value>host1,host2</value>
  <description>Allow the flume user to connect only from host1 and host2 to impersonate a
user</description>
</property>
```

Flume configuration for multiple sinks

```
agent.sinks.sink-1.type = HDFS
agent.sinks.sink-1.hdfs.kerberosPrincipal = flume-hostid/_HOST@REALM
agent.sinks.sink-1.hdfs.kerberosKeytab = /etc/security/phd/keytab/flume.service.keytab
agent.sinks.sink-1.hdfs.proxyUser = log1

agent.sinks.sink-2.type = HDFS
agent.sinks.sink-2.hdfs.kerberosPrincipal = flume-hostid/_HOST@REALM
agent.sinks.sink-2.hdfs.kerberosKeytab = /etc/security/phd/keytab/flume.service.keytab
agent.sinks.sink-2.hdfs.proxyUser = log2
```

Flume configuration example:

```
agent.sources = r1
agent.sinks = k1
agent.channels = c1

# Describe/configure the source
agent.sources.r1.type = netcat
agent.sources.r1.bind = localhost
agent.sources.r1.port = 44444

# Describe the sink
agent.sinks.k1.type = hdfs
agent.sinks.k1.hdfs.path = hdfs://centos64-1.localdomain/user/flume1
agent.sinks.k1.hdfs.fileType = DataStream
agent.sinks.k1.hdfs.kerberosPrincipal = flume/_HOST@REALM
agent.sinks.k1.hdfs.kerberosKeytab = /etc/security/phd/keytab/flume.service.keytab
agent.sinks.k1.hdfs.proxyUser = log1

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

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

agent.sources = r2
agent.sinks = k2
agent.channels = c2

# Describe/configure the source
agent.sources.r2.type = netcat
agent.sources.r2.bind = localhost
agent.sources.r2.port = 55555

# Describe the sink
agent.sinks.k2.type = hdfs
agent.sinks.k2.hdfs.path = hdfs://centos64-1.localdomain/user/flume2
```

```
agent.sinks.k2.hdfs.fileType = DataStream
agent.sinks.k2.hdfs.kerberosPrincipal = flume/_HOST@REALM
agent.sinks.k2.hdfs.kerberosKeytab = /etc/security/phd/keytab/flume.service.keytab
agent.sinks.k2.hdfs.proxyUser = log2

# Use a channel which buffers events in memory
agent.channels.c2.type = memory
agent.channels.c2.capacity = 1000
agent.channels.c2.transactionCapacity = 100

# Bind the source and sink to the channel
agent.sources.r2.channels = c2
agent.sinks.k2.channel = c2
```

Oozie Security Configuration

This section describes Oozie security configuration.

Prerequisites

- The Oozie server must be installed on the cluster.
- Security must be enabled for HDFS and yarn on the cluster.

Create the Oozie Principal

On the KDC admin server, create a principal for the Oozie server:

```
kadmin.local: addprinc -randkey oozie/<host_fqdn>@REALM
```

Where <host_fqdn> is the host where the Oozie server is running.

Create the HTTP Principal for the Oozie Server



Skip this step if the principal has already been created.

On the KDC admin server, create the HTTP principal for the host running the Oozie server. This principal may have been created when enabling security for other services. Skip this step if that is the case.

```
kadmin.local: addprinc -randkey HTTP/<host_fqdn>@REALM
```

Create the Oozie Keytab Files

On the KDC admin server, create the Oozie keytab files:

```
kadmin.local: ktadd -norandkey -k /etc/security/phd/keytab/oozie-<host_fqdn>.service.keytab  
oozie/<host_fqdn>@REALM HTTP/<host_fqdn>@REALM
```

Copy the Oozie Keytab Files to the Oozie server and change the ownership and permission

Move the keytab file created in the previous step to the /etc/security/phd/keytab directory on the host running the Oozie server:

```
scp /etc/security/phd/keytab/oozie-<host_fqdn>.service.keytab
<oozie.host.name>:/etc/security/phd/keytab/
```

Then run the following commands on the Oozie server:

```
cd /etc/security/phd/keytab

chgrp hadoop oozie-<host_fqdn>.service.keytab

chown oozie oozie-<host_fqdn>.service.keytab

chmod 400 oozie-<host_fqdn>.service.keytab

ln -s oozie-<host_fqdn>.service.keytab oozie.service.keytab
```

Edit the Oozie Configuration

On the Oozie server, locate the Oozie configuration file, `/etc/gphd/oozie/conf/oozie-site.xml`, and change the following properties to configure Oozie to run in secure mode:

Property	Value
oozie.service.HadoopAccessorService.kerberos.enabled	true
local.realm	<YOUR-REALM>
oozie.service.HadoopAccessorService.keytab.file	/etc/security/phd/keytab/oozie.serv
oozie.service.HadoopAccessorService.kerberos.principal	oozie/_HOST@<YOUR-REALM>
oozie.authentication.type	simple
oozie.authentication.kerberos.principal	HTTP/_HOST@<YOUR-REALM>
oozie.authentication.kerberos.name.rules	Use the value configured for hadoop.security.auth_to_local in core The default value is DEFAULT if not set in the core-site.xml.

Using Oozie with a Secure Hive Metastore Server

For hive actions to connect to a secure hive metastore, you need to add credential configuration to the `/etc/gphd/oozie/conf/oozie-site.xml` file, as follows:

```
<property>
  <name>oozie.credentials.credentialclasses</name>
  <value>

hcat=org.apache.oozie.action.hadoop.HCatCredentials,hive=org.apache.oozie.action.hadoop.HbaseCredenti
</value>
</property>
```

Verify Secure Oozie

Login as the authorized user, `kinit` then `cd` into the home directory of the authorized user, then run the Oozie Hive action:

```
oozie job -oozie http://<oozier_hostname>:11000/oozie -config examples/apps/hive/job.properties  
-run
```

Check Oozie job status using the `job_ID` returned after running the above command:

```
oozie job -oozie http://<oozie.host.name>:11000/oozie -info <JOBID>
```

Sqoop Security Configuration

`kerberos` configuration is required for user submitting sqoop job.

Pig Security Configuration

`kerberos` configuration is required for user submitting pig job.

Mahout Security Configuration

`kerberos` configuration is required for user submitting mahout job.

Troubleshooting

Log Files

A good first step is to look for exceptions that may give you a clue as to the problem in the log files (where `hostname` is the host where the log file is located):

- **namenode:** `/var/log/gphd/hadoop-hdfs/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/hadoop-hdfs-datanode-hostname.log`
- **nodemanager:** `/var/log/gphd/hadoop-yarn/yarn-yarn-nodemanager-hostname.log`

You can enable debug level logging for the Java Kerberos classes by editing `/etc/default/hadoop` and setting the following:

```
HADOOP_OPTS="$HADOOP_OPTS -Dsun.security.krb5.debug=true"
```

Also remember that Kerberos operation in Hadoop is very sensitive to proper networking configuration:

- Host IP's for service nodes must reverse map to the FQDN's used to create the node principal for the service/FQDN
- `hostname -f` on a node must give the FQDN used to create the principal for the service/FQDN

Make sure your networking is properly configured before attempting to secure a cluster!

Data node will not start:

- If you are getting a message about data node requiring privileged resources to start check your ports 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 run `kinit -k -t /etc/security/phd/keytab/servicename.service.keytab;` you should get no errors (just a prompt back). If there is an error check that the principal and keytab are correct. Check to make sure you used `-norandkey` when creating keytab files.

Node manager will not start:

- Login failure due to policy error exceptions in logs (typically seen as a remote exception to node manager for resource manager): check `/usr/lib/gphd/hadoop/etc/hadoop/hadoop-policy.xml` and replace any occurrences of `${HADOOP_HDFS_USER}` with `hdfs` and `${HADOOP_YARN_USER}` with `yarn`.