

Hortonworks Data Platform

Hadoop Security Guide

(Oct 28, 2014)

Hortonworks Data Platform : Hadoop Security Guide

Copyright © 2012-2014 Hortonworks, Inc. Some rights reserved.

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, Zookeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, [training](#) and partner-enablement services. All of our technology is, and will remain free and open source.

Please visit the [Hortonworks Data Platform](#) page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the [Support](#) or [Training](#) page. Feel free to [Contact Us](#) directly to discuss your specific needs.



Except where otherwise noted, this document is licensed under
Creative Commons Attribution ShareAlike 3.0 License.
<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

Table of Contents

1. Hadoop Security Features	1
2. Set up Authentication for Hadoop Cluster Components	3
2.1. Setting Up Security for Manual Installs	3
2.1.1. Preparing Kerberos	3
2.1.2. Installing and Configuring the KDC	4
2.1.3. Creating the Database and Setting Up the First Administrator	5
2.1.4. Creating Service Principals and Keytab Files for HDP	6
2.2. Configuring HDP	9
2.2.1. Configuration Overview	9
2.2.2. Creating Mappings Between Principals and UNIX Usernames	9
2.2.3. Adding Security Information to Configuration Files	11
2.3. Configure Secure HBase and ZooKeeper	27
2.3.1. Configure HBase Master	27
2.3.2. Create JAAS configuration files	29
2.3.3. Start HBase and ZooKeeper services	31
2.3.4. Configure secure client side access for HBase	32
2.3.5. Optional: Configure client-side operation for secure operation - Thrift Gateway	33
2.3.6. Optional: Configure client-side operation for secure operation - REST Gateway	33
2.3.7. Configure HBase for Access Control Lists (ACL)	34
2.4. Setting up One-Way Trust with Active Directory	35
2.4.1. Configure Kerberos Hadoop Realm on the AD DC	35
2.4.2. Configure the AD Domain on the KDC and Hadoop Cluster Hosts	36
2.5. Allowing Impersonation	37
3. Data Protection	38
3.1. Enable RPC Encryption for the Hadoop Cluster	39
3.2. Enable Data Transfer Protocol	39
3.3. Enable SSL on HDP Components	39
3.3.1. Understanding Hadoop SSL Keystore Factory	40
3.3.2. Manage SSL Certificates	42
3.3.3. Enable SSL for WebHDFS, MapReduce Shuffle, and YARN	49
3.3.4. Enable SSL on Oozie	52
3.3.5. Enable SSL on WebHBase and the HBase REST API	53
3.3.6. Enable SSL on HiveServer2	54
3.4. Connect to SSL Enabled Components	55
3.4.1. Connect to SSL Enabled HiveServer2 using JDBC	55
3.4.2. Connect to SSL Enabled Oozie Server	55

List of Tables

2.1. Service Principals	7
2.2. Service Keytab File Names	8
2.3. core-site.xml	11
2.4. core-site.xml	12
2.5. core-site.xml	12
2.6. hdfs-site.xml	13
2.7. yarn-site.xml	17
2.8. mapred-site.xml	19
2.9. hbase-site.xml	23
2.10. hive-site.xml	25
2.11. oozie-site.xml	26
2.12. webhcat-site.xml	27
3.1. Configure Data Protection for HDP Components	39
3.2. Components that Support SSL	40
3.3. Configuration Properties in ssl-server.xml	50

1. Hadoop Security Features

For organizations that store sensitive data in the Hadoop ecosystem, such as proprietary or personal data that is subject to regulatory compliance (HIPPA, PCI, DSS, FISAM, etc), security is essential. Many organizations also have to adhere to strict internal security policies.

The Hortonworks Data Platform provides a comprehensive approach to security in the following key areas:

- **Perimeter security:** HDP enables isolation of the Hadoop cluster using a gateway and properly configured firewall rules. HDP supports the following perimeter security:
 - [Apache Knox Gateway](#)
 - Gateway clients
- **Authentication:** HDP provides single authentication point for services and users that integrates with existing enterprise identity and access management systems. HDP supports the following authentication services:
 - Kerberos
 - LDAP
 - Local Unix System
 - [SSO](#) (at the perimeter through Apache Knox Gateway)
- **Authorization (Access Control):** HDP provides features that allow system administrators to control access to Hadoop data using role-based authorization. HDP supports the following authorization models:
 - Fine-grained access control for data stored in HDFS
 - Resource-level access control for YARN
 - Coarser-grain service level access control for MapReduce Operations
 - Table and column family level access control for HBase data, and extended ACLs for cell level control with Accumulo.
 - Table level access control for Apache Hive data sets
- **Accounting (Security auditing and monitoring):** HDP allows you to track Hadoop activity using Native Auditing (audit logs), [perimeter security auditing logs](#) on the Knox Gateway, and from a central location, the [HDP Security Administration console](#), including:
 - Access requests
 - Data processing operations
 - Data changes

- **Data Protection:** HDP provides the mechanisms for encrypting data in flight, and requires the use of partner solutions for encrypting data at rest, data discovery, and data masking. HDP supports the following wire encryption methods:
 - [SSL for HDP Components](#)
 - [RPC encryption](#)
 - [Data Transfer Protocol](#)

2. Set up Authentication for Hadoop Cluster Components

Authentication: HDP provides single authentication point for services and users that integrates with existing enterprise identity and access management systems. HDP Supports the following authentication services:

- Kerberos
- LDAP
- Local Unix System
- [SSO](#) (at the perimeter through Apache Knox Gateway)

2.1. Setting Up Security for Manual Installs

This section provides information on enabling security for a manually installed version of HDP.

2.1.1. Preparing Kerberos

To create secure communication among its various components, HDP uses Kerberos. Kerberos is a third-party authentication mechanism, in which users and services that users wish to access rely on a the Kerberos server to authenticate each to the other. This mechanism also supports encrypting all traffic between the user and the service. The Kerberos server itself is known as the *Key Distribution Center*, or KDC. At a high level, it has the following parts:

- A database of the users and services (known as *principals*) that it knows about and their respective Kerberos passwords
- An *authentication server (AS)* which performs the initial authentication and issues a *Ticket Granting Ticket (TGT)*
- A *Ticket Granting Server (TGS)* that issues subsequent service tickets based on the initial TGT.

A user principal requests authentication from the AS. The AS returns a TGT that is encrypted using the user principal's Kerberos password, which is known only to the user principal and the AS. The user principal decrypts the TGT locally using its Kerberos password, and from that point forward, until the ticket expires, the user principal can use the TGT to get service tickets from the TGS.

Because a service principal cannot provide a password each time to decrypt the TGT, it uses a special file, called a *keytab*, which contains its authentication credentials.

The service tickets are what allow the principal to access various services. The set of hosts, users, and services over which the Kerberos server has control is called a *realm*.



Note

Because Kerberos is a time-sensitive protocol, all hosts in the realm must be time-synchronized, for example, by using the Network Time Protocol (NTP). If the local system time of a client differs from that of the KDC by as little as 5 minutes (the default), the client will not be able to authenticate.

2.1.2. Installing and Configuring the KDC

To use Kerberos with HDP, either use an existing KDC or install a new one for HDP only. The following gives a very high level description of the installation process. For more information, see [RHEL documentation](#), [CentOS documentation](#), [SLES documentation](#), or [Ubuntu and Debian documentation](#).

1. Install the KDC server:

- On RHEL, CentOS, or Oracle Linux, run:

```
yum install krb5-server krb5-libs krb5-auth-dialog krb5-workstation
```

- On SLES, run:

```
zypper install krb5 krb5-server krb5-client
```

- On Ubuntu or Debian, run:

```
apt-get install krb5 krb5-server krb5-client
```



Note

The host on which you install the KDC must itself be secure.

When the server is installed you must edit the two main configuration files, located by default here:

2. Update the KDC configuration by replacing `EXAMPLE.COM` with your domain and `kerberos.example.com` with the FQDN of the KDC host; the configuration files are located:

- On RHEL, CentOS, or Oracle Linux:

- `/etc/krb5.conf`
- `/var/kerberos/krb5kdc/kdc.conf`.

- On SLES:

- `/etc/krb5.conf`
- `/var/lib/kerberos/krb5kdc/kdc.conf`

- On Ubuntu or Debian:

- `/etc/krb5.conf`

- `/var/kerberos/krb5kdc/kdc.conf`.

3. Copy the updated `krb5.conf` to every cluster node.

2.1.3. Creating the Database and Setting Up the First Administrator

1. Use the utility `kdb5_util` to create the Kerberos database:

- On RHEL, CentOS, or Oracle Linux:

```
/usr/sbin/kdb5_util create -s
```

- On SLES:

```
[on SLES]
kdb5_util create -s
```

- On Ubuntu or Debian:

```
[on Ubuntu or Debian]
kdb5_util -s create
```



Note

The `-s` option stores the master server key for the database in a *stash* file. If the stash file is not present, you must log into the KDC with the master password (specified during installation) each time it starts. This will automatically regenerate the master server key.

2. Set up the KDC Access Control List (ACL):

- On RHEL, CentOS, or Oracle Linux add administrators to `/var/kerberos/krb5kdc/kadm5.acl`.
- On SLES, add administrators to `/var/lib/kerberos/krb5kdc/kadm5.acl`.



Note

For example, the following line grants full access to the database for users with the `admin` extension:

```
*/admin@EXAMPLE.COM *
```

3. Restart `kadmin` for the change to take effect.

4. Create the first user principal. This must be done at a terminal window on the KDC machine itself, while you are logged in as `root`. Notice the `.local`. Normal `kadmin` usage requires that a principal with appropriate access already exist. The `kadmin.local` command can be used even if no principals exist.

```
/usr/sbin/kadmin.local -q "addprinc $username/admin"
```

Now this user can create additional principals either on the KDC machine or through the network. The following instruction assume you are using the KDC machine.

5. On the KDC, start Kerberos:

- On RHEL, CentOS, or Oracle Linux:

```
/sbin/service krb5kdc start
/sbin/service kadmind start
```

- On SLES:

```
rckrb5kdc start
rckadmind start
```

- On Ubuntu or Debian:

```
/etc/init.d/krb5-kdc start
/etc/init.d/kadmind start
```

2.1.4. Creating Service Principals and Keytab Files for HDP

Each service in HDP must have its own principal. As services do not login with a password to acquire their tickets, their principal's authentication credentials are stored in a keytab file, which is extracted from the Kerberos database and stored locally with the service principal.

First you must create the principal, using mandatory naming conventions. Then you must create the keytab file with that principal's information and copy the file to the keytab directory on the appropriate service host.

Step 1: Create a service principal using the `kadmin` utility:

```
kadmin: addprinc -randkey $principal_name/$service-host-FQDN@$hadoop.realm
```

You must have a principal with administrative permissions to use this command. The `randkey` is used to generate the password.

where



Note

In the example each service principal's name has appended to it the fully qualified domain name of the host on which it is running. This is to provide a unique principal name for services that run on multiple hosts, like DataNodes and TaskTrackers. The addition of the hostname serves to distinguish, for example, a request from DataNode A from a request from DataNode B. This is important for two reasons:

- If the Kerberos credentials for one DataNode are compromised, it does not automatically lead to all DataNodes being compromised
- If multiple DataNodes have exactly the same principal and are simultaneously connecting to the NameNode, and if the Kerberos authenticator being sent

happens to have same timestamp, then the authentication would be rejected as a replay request.

The `$principal_name` part of the name must match the values in the table below.



Note

The NameNode, Secondary NameNode, and Oozie require two principals each.



Note

If you are configuring High Availability (HA) for a Quorum-based NameNode, you must also generate a principle (`jn/$FQDN`) and keytab (`jn.service.keytab`) for each JournalNode. JournalNode also requires the keytab for its HTTP service. If the JournalNode is deployed on the same host as a NameNode, the same keytab file (`spnego.service.keytab`) can be used for both. In addition, HA requires two NameNodes. Both the active and standby NameNode require their own principle and keytab files. The service principles of the two NameNodes can share the same name, specified with the `dfs.namenode.kerberos.principal` property in `hdfs-site.xml`, but the NameNodes still have different fully qualified domain names.

Table 2.1. Service Principals

Service	Component	Mandatory Principal Name
HDFS	NameNode	<code>nn/\$FQDN</code>
HDFS	NameNode HTTP	<code>HTTP/\$FQDN</code>
HDFS	SecondaryNameNode	<code>nn/\$FQDN</code>
HDFS	SecondaryNameNode HTTP	<code>HTTP/\$FQDN</code>
HDFS	DataNode	<code>dn/\$FQDN</code>
MR2	History Server	<code>jhs/\$FQDN</code>
MR2	History Server HTTP	<code>HTTP/\$FQDN</code>
YARN	ResourceManager	<code>rm/\$FQDN</code>
YARN	NodeManager	<code>nm/\$FQDN</code>
Oozie	Oozie Server	<code>oozie/\$FQDN</code>
Oozie	Oozie HTTP	<code>HTTP/\$FQDN</code>
Hive	Hive Metastore	<code>hive/\$FQDN</code>
	HiveServer2	
Hive	WebHCat	<code>HTTP/\$FQDN</code>
HBase	MasterServer	<code>hbase/\$FQDN</code>
HBase	RegionServer	<code>hbase/\$FQDN</code>
Hue	Hue Interface	<code>hue/\$FQDN</code>
ZooKeeper	ZooKeeper	<code>zookeeper/\$FQDN</code>
Nagios Server	Nagios	<code>nagios/\$FQDN</code>
JournalNode Server ^a	JournalNode	<code>jn/\$FQDN</code>
Gateway	Knox	<code>knox/\$FQDN</code>

^aOnly required if you are setting up NameNode HA.

For example: To create the principal for a DataNode service, issue this command:

```
kadmin: addprinc -randkey dn/$datanode-host@$hadoop.realm
```

Step 2: Extract the related keytab file and place it in the keytab directory (by default /etc/krb5.keytab) of the appropriate respective components:

```
kadmin: xst -k $keytab_file_name $principal_name/fully.qualified.domain.name
```

You must use the mandatory names for the `$keytab_file_name` variable shown in this table.

Table 2.2. Service Keytab File Names

Component	Principal Name	Mandatory Keytab File Name
NameNode	nn/\$FQDN	nn.service.keytab
NameNode HTTP	HTTP/\$FQDN	spnego.service.keytab
SecondaryNameNode	nn/\$FQDN	nn.service.keytab
SecondaryNameNode HTTP	HTTP/\$FQDN	spnego.service.keytab
DataNode	dn/\$FQDN	dn.service.keytab
MR2 History Server	jhs/\$FQDN	nm.service.keytab
MR2 History Server HTTP	HTTP/\$FQDN	spnego.service.keytab
YARN	rm/\$FQDN	rm.service.keytab
YARN	nm/\$FQDN	nm.service.keytab
Oozie Server	oozie/\$FQDN	oozie.service.keytab
Oozie HTTP	HTTP/\$FQDN	spnego.service.keytab
Hive Metastore	hive/\$FQDN	hive.service.keytab
HiveServer2		
WebHCat	HTTP/\$FQDN	spnego.service.keytab
HBase Master Server	hbase/\$FQDN	hbase.service.keytab
HBase RegionServer	hbase/\$FQDN	hbase.service.keytab
Hue	hue/\$FQDN	hue.service.keytab
ZooKeeper	zookeeper/\$FQDN	zk.service.keytab
Nagios Server	nagios/\$FQDN	nagios.service.keytab
Journal Server ^a	jn/\$FQDN	jn.service.keytab
Knox Gateway ^b	knox/\$FQDN	knox.service.keytab

^aOnly required if you are setting up NameNode HA.

^bOnly required if you are using a Knox Gateway.

For example: To create the keytab files for the NameNode, issue these commands:

```
kadmin: xst -k nn.service.keytab nn/$namenode-host
kadmin: xst -k spnego.service.keytab HTTP/$namenode-host
```

When you have created the keytab files, copy them to the keytab directory of the respective service hosts.

Step 3: Verify that the correct keytab files and principals are associated with the correct service using the `klist` command. For example, on the NameNode:

```
klist -k -t /etc/security/nn.service.keytab
```

Do this on each respective service in your cluster.

2.2. Configuring HDP

This section provides information on configuring HDP for Kerberos.

- [Configuration Overview](#)
- [Creating Mappings Between Principals and UNIX Usernames](#)
- [Creating the Database and Setting Up the First Administrator](#)
- [Creating Principals and Keytab Files for HDP](#)

2.2.1. Configuration Overview

Configuring HDP for Kerberos has two parts:

- Creating a mapping between service principals and UNIX usernames.

Hadoop uses group memberships of users at various places, such as to determine group ownership for files or for access control.

A user is mapped to the groups it belongs to using an implementation of the `GroupMappingServiceProvider` interface. The implementation is pluggable and is configured in `core-site.xml`.

By default Hadoop uses `ShellBasedUnixGroupsMapping`, which is an implementation of `GroupMappingServiceProvider`. It fetches the group membership for a username by executing a UNIX shell command. In secure clusters, since the usernames are actually Kerberos principals, `ShellBasedUnixGroupsMapping` will work only if the Kerberos principals map to valid UNIX usernames. Hadoop provides a feature that lets administrators specify mapping rules to map a Kerberos principal to a local UNIX username .

- Adding information to three main service configuration files.

There are several optional entries in the three main service configuration files that must be added to enable security on HDP.

2.2.2. Creating Mappings Between Principals and UNIX Usernames

HDP uses a rule-based system to create mappings between service principals and their related UNIX usernames. The rules are specified in the `core-site.xml` configuration file as the value to the optional key `hadoop.security.auth_to_local`.

The default rule is simply named `DEFAULT`. It translates all principals in your default domain to their first component. For example, `myusername@APACHE.ORG` and `myusername/admin@APACHE.ORG` both become `myusername`, assuming your default domain is `APACHE.ORG`.

2.2.2.1. Creating Rules

To accommodate more complex translations, you can create a hierarchical set of rules to add to the default. Each rule is divided into three parts: base, filter, and substitution.

2.2.2.1.1. The Base

The base begins with the number of components in the principal name (excluding the realm), followed by a colon, and the pattern for building the username from the sections of the principal name. In the pattern section \$0 translates to the realm, \$1 translates to the first component and \$2 to the second component.

For example:

```
[1:$1@$0] translates myusername@APACHE.ORG to myusername@APACHE.ORG
```

```
[2:$1] translates myusername/admin@APACHE.ORG to myusername
```

```
[2:$1%$2] translates myusername/admin@APACHE.ORG to "myusername%admin
```

2.2.2.1.2. The Filter

The filter consists of a regex in a parentheses that must match the generated string for the rule to apply.

For example:

```
(.*%admin) matches any string that ends in %admin
```

```
(.*@SOME.DOMAIN) matches any string that ends in @SOME.DOMAIN
```

2.2.2.1.3. The Substitution

The substitution is a `sed` rule that translates a regex into a fixed string.

For example:

```
s/@ACME\ .COM// removes the first instance of @SOME.DOMAIN.
```

```
s/[A-Z]*\ .COM// removes the first instance of @ followed by a name followed by COM.
```

```
s/X/Y/g replaces all of the X in the name with Y
```

2.2.2.2. Examples

- If your default realm was `APACHE.ORG`, but you also wanted to take all principals from `ACME.COM` that had a single component `joe@ACME.COM`, you would create this rule:

```
RULE:[1:$1@$0](.@ACME.COM)s/@.//
DEFAULT
```

- To also translate names with a second component, you would use these rules:

```
RULE:[1:$1@$0](.@ACME.COM)s/@.//
RULE:[2:$1@$0](.@ACME.COM)s/@.//
DEFAULT
```

- To treat all principals from `APACHE.ORG` with the extension `/admin` as `admin`, your rules would look like this:

```
RULE[2:$1%$2@$0](.admin@APACHE.ORG)s/.admin/
DEFAULT
```

2.2.3. Adding Security Information to Configuration Files

To enable security on HDP, you must add optional information to various configuration files.

Before you begin, set `JSVC_Home` in `hadoop-env.sh`.

- For RHEL/CentOS/Oracle Linux:

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

- For SLES and Ubuntu:

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

2.2.3.1. core-site.xml

To the `core-site.xml` file on every host in your cluster, you must add the following information:

Table 2.3. core-site.xml

Property Name	Property Value	Description
<code>hadoop.security.authentication</code>	<code>kerberos</code>	Set the authentication type for the cluster. Valid values are: <code>simple</code> or <code>kerberos</code> .
<code>hadoop.rpc.protection</code>	<code>authentication; integrity; privacy</code>	This is an [OPTIONAL] setting. If not set, defaults to <code>authentication</code> . <code>authentication</code> = authentication only; the client and server mutually authenticate during connection setup. <code>integrity</code> = authentication and integrity; guarantees the integrity of data exchanged between client and server as well as authentication. <code>privacy</code> = authentication, integrity, and confidentiality; guarantees that data exchanged between client and server is encrypted and is not readable by a “man in the middle”.
<code>hadoop.security.authorization</code>	<code>true</code>	Enable authorization for different protocols.
<code>hadoop.security.auth_to_local</code>	The mapping rules. For example: <code>RULE:[2:\$1@\$0]</code> <code>([jt]t@.*EXAMPLE.COM)s/.*/</code> <code>mapred/ RULE:[2:\$1@\$0]</code> <code>([nd]n@.*EXAMPLE.COM)s/.*/</code> <code>hdfs/ RULE:[2:\$1@\$0]</code> <code>(hm@.*EXAMPLE.COM)s/.*/</code> <code>hbase/ RULE:[2:\$1@\$0]</code> <code>(rs@.*EXAMPLE.COM)s/.*/</code> <code>hbase/ DEFAULT</code>	The mapping from Kerberos principal names to local OS user names. See Creating Mappings Between Principals and UNIX Usernames for more information.

The XML for these entries:

```

<property>
  <name>hadoop.security.authentication</name>
  <value>kerberos</value>
  <description>    Set the
authentication for the cluster. Valid values are: simple or
kerberos.
  </description>
</property>

<property>
  <name>hadoop.security.authorization</name>
  <value>true</value>
  <description>    Enable
authorization for different protocols.
  </description>
</property>

<property>

  <name>hadoop.security.auth_to_local</name>
  <value>
RULE:[2:$1@$0]([jt]t@.*EXAMPLE.COM)s/./mapred/
RULE:[2:$1@$0]([nd]n@.*EXAMPLE.COM)s/./hdfs/
RULE:[2:$1@$0](hm@.*EXAMPLE.COM)s/./hbase/
RULE:[2:$1@$0](rs@.*EXAMPLE.COM)s/./hbase/
DEFAULT</value>
  <description>The mapping from kerberos principal names
to local OS user names.</description>
</property>

```

When using the Knox Gateway, add the following to the `core-site.xml` file on the *master nodes* host in your cluster:

Table 2.4. core-site.xml

Property Name	Property Value	Description
<code>hadoop.proxyuser.knox.groups</code>	<code>users</code>	Grants proxy privileges for Knox user.
<code>hadoop.proxyuser.knox.hosts</code>	<code>\$knox_host_FQDN</code>	Identifies the Knox Gateway host.

When using the Hue, add the following to the `core-site.xml` file on the *master nodes* host in your cluster:

Table 2.5. core-site.xml

Property Name	Property Value	Description
<code>hue.kerberos.principal.shortname</code>	<code>hue</code>	Hue Kerberos principal name.
<code>hadoop.proxyuser.hue.groups</code>	<code>*</code>	Group to which all the Hue users belong. Use the wild card character to select multiple groups, for example <code>cli*</code> .
<code>hadoop.proxyuser.hue.hosts</code>	<code>*</code>	
<code>hadoop.proxyuser.knox.hosts</code>	<code>\$hue_host_FQDN</code>	Identifies the Knox Gateway host.

The XML for these entries:

```
<property>
  <name>hadoop.security.authentication</name>
  <value>kerberos</value>
  <description>    Set the
authentication for the cluster. Valid values are: simple or
kerberos.
  </description>
</property>

<property>
  <name>hadoop.security.authorization</name>
  <value>>true</value>
  <description>    Enable
authorization for different protocols.
  </description>
</property>

<property>

  <name>hadoop.security.auth_to_local</name>
  <value>
RULE:[2:$1@$0]([jt]t@.*EXAMPLE.COM)s/.*/mapred/
RULE:[2:$1@$0]([nd]n@.*EXAMPLE.COM)s/.*/hdfs/
RULE:[2:$1@$0](hm@.*EXAMPLE.COM)s/.*/hbase/
RULE:[2:$1@$0](rs@.*EXAMPLE.COM)s/.*/hbase/
DEFAULT</value> <description>The mapping from kerberos principal names
to local OS user names.</description>
</property>
<property>
  <name>hadoop.proxyuser.knox.groups</name>
  <value>users</value>
</property>
<property>
  <name>hadoop.proxyuser.knox.hosts</name>
  <value>Knox.EXAMPLE.COM</value>
</property>
```

2.2.3.2. hdfs-site.xml

To the `hdfs-site.xml` file on every host in your cluster, you must add the following information:

Table 2.6. hdfs-site.xml

Property Name	Property Value	Description
<code>dfs.permissions.enabled</code>	<code>true</code>	If <code>true</code> , permission checking in HDFS is enabled. If <code>false</code> , permission checking is turned off, but all other behavior is unchanged. Switching from one parameter value to the other does not change the mode, owner or group of files or directories.
<code>dfs.permissions.supergroup</code>	<code>hdfs</code>	The name of the group of super-users.
<code>dfs.block.access.token.enable</code>	<code>true</code>	If <code>true</code> , access tokens are used as capabilities for accessing DataNodes. If <code>false</code> , no access tokens are checked on accessing DataNodes.

Property Name	Property Value	Description
dfs.namenode.kerberos.principal	nn/_HOST@EXAMPLE.COM	Kerberos principal name for the NameNode.
dfs.secondary.namenode.kerberos.principal	ns/_HOST@EXAMPLE.COM	Kerberos principal name for the secondary NameNode.
dfs.web.authentication.kerberos.principal	HTTP/_HOST@EXAMPLE.COM	The HTTP Kerberos principal used by Hadoop-Auth in the HTTP endpoint. The HTTP Kerberos principal MUST start with 'HTTP/' per Kerberos HTTP SPNEGO specification.
dfs.web.authentication.kerberos.keytab	/etc/security/keytabs/spnego.service.keytab	The Kerberos keytab file with the credentials for the HTTP Kerberos principal used by Hadoop-Auth in the HTTP endpoint.
dfs.datanode.kerberos.principal	nn/_HOST@EXAMPLE.COM	The Kerberos principal that the DataNode runs as. "_HOST" is replaced by the real host name .
dfs.namenode.keytab.file	/etc/security/keytabs/nn.service.keytab	Combined keytab file containing the NameNode service and host principals.
dfs.secondary.namenode.keytab.file	/etc/security/keytabs/nn.service.keytab	Combined keytab file containing the NameNode service and host principals. <question?>
dfs.datanode.keytab.file	/etc/security/keytabs/dn.service.keytab	The filename of the keytab file for the DataNode.
dfs.https.port	50470	The HTTPS port to which the NameNode binds
dfs.namenode.https-address	Example: ip-10-111-59-170.ec2.internal:50470	The HTTPS address to which the NameNode binds
dfs.datanode.data.dir.perm	750	The permissions that must be set on the dfs.data.dir directories. The DataNode will not come up if all existing dfs.data.dir directories do not have this setting. If the directories do not exist, they will be created with this permission
dfs.cluster administrators	hdfs	ACL for who all can view the default servlets in the HDFS
dfs.namenode.kerberos.internal.spnego.principal	dfswebauthn/_HOST@EXAMPLE.COM	Internal SPNEGO principal for NameNode
dfs.secondary.namenode.kerberos.internal.spnego.principal	dfswebauthn/_HOST@EXAMPLE.COM	Internal SPNEGO principal for secondary NameNode

The XML for these entries:

```
<property>
  <name>dfs.permissions</name>
  <value>true</value>
  <description> If "true", enable permission checking in
HDFS. If "false", permission checking is turned
off, but all other behavior is
unchanged. Switching from one parameter value to the other does
not change the mode, owner or group of files or
directories. </description>
</property>

<property>
  <name>dfs.permissions.supergroup</name>
  <value>hdfs</value>
  <description>The name of the group of
```

```
        super-users.</description>
</property>

<property>
  <name>dfs.namenode.handler.count</name>
  <value>100</value>
  <description>Added to grow Queue size so that more
  client connections are allowed</description>
</property>

<property>
  <name>ipc.server.max.response.size</name>
  <value>5242880</value>
</property>

<property>
  <name>dfs.block.access.token.enable</name>
  <value>true</value>
  <description> If "true", access tokens are used as capabilities
  for accessing datanodes. If "false", no access tokens are checked on
  accessing datanodes. </description>
</property>

<property>
  <name>dfs.namenode.kerberos.principal</name>
  <value>nn/_HOST@EXAMPLE.COM</value>
  <description> Kerberos principal name for the
  NameNode </description>
</property>

<property>
  <name>dfs.secondary.namenode.kerberos.principal</name>
  <value>nn/_HOST@EXAMPLE.COM</value>
  <description>Kerberos principal name for the secondary NameNode.
  </description>
</property>

<property>
  <!--cluster variant -->
  <name>dfs.secondary.http.address</name>
  <value>ip-10-72-235-178.ec2.internal:50090</value>
  <description>Address of secondary namenode web server</description>
</property>

<property>
  <name>dfs.secondary.https.port</name>
  <value>50490</value>
  <description>The https port where secondary-namenode
  binds</description>
</property>

<property>
  <name>dfs.web.authentication.kerberos.principal</name>
  <value>HTTP/_HOST@EXAMPLE.COM</value>
  <description> The HTTP Kerberos principal used by Hadoop-Auth in the
  HTTP endpoint.
  The HTTP Kerberos principal MUST start with 'HTTP/' per Kerberos HTTP
  SPNEGO specification.
  </description>
</property>
```

```
<property>
  <name>dfs.web.authentication.kerberos.keytab</name>
  <value>/etc/security/keytabs/spnego.service.keytab</value>
  <description>The Kerberos keytab file with the credentials for the
  HTTP
  Kerberos principal used by Hadoop-Auth in the HTTP endpoint.
  </description>
</property>

<property>
  <name>dfs.datanode.kerberos.principal</name>
  <value>dn/_HOST@EXAMPLE.COM</value>
  <description>
  The Kerberos principal that the DataNode runs as. "_HOST" is replaced
  by the real
  host name.
  </description>
</property>

<property>
  <name>dfs.namenode.keytab.file</name>
  <value>/etc/security/keytabs/nn.service.keytab</value>
  <description>
  Combined keytab file containing the namenode service and host
  principals.
  </description>
</property>

<property>
  <name>dfs.secondary.namenode.keytab.file</name>
  <value>/etc/security/keytabs/nn.service.keytab</value>
  <description>
  Combined keytab file containing the namenode service and host
  principals.
  </description>
</property>

<property>
  <name>dfs.datanode.keytab.file</name>
  <value>/etc/security/keytabs/dn.service.keytab</value>
  <description>
  The filename of the keytab file for the DataNode.
  </description>
</property>

<property>
  <name>dfs.https.port</name>
  <value>50470</value>
  <description>The https port where namenode
  binds</description>
</property>

<property>
  <name>dfs.https.address</name>
  <value>ip-10-111-59-170.ec2.internal:50470</value>
  <description>The https address where namenode binds</description>
</property>

<property>
```

```

<name>dfs.datanode.data.dir.perm</name>
<value>750</value>
<description>The permissions that should be there on
dfs.data.dir directories. The datanode will not come up if the
permissions are different on existing dfs.data.dir directories. If
the directories don't exist, they will be created with this
permission.</description>
</property>

<property>
  <name>dfs.access.time.precision</name>
  <value>0</value>
  <description>The access time for HDFS file is precise upto this
value.The default value is 1 hour. Setting a value of 0
disables access times for HDFS.
</description>
</property>

<property>
  <name>dfs.cluster.administrators</name>
  <value> hdfs</value>
  <description>ACL for who all can view the default
servlets in the HDFS</description>
</property>

<property>
  <name>ipc.server.read.threadpool.size</name>
  <value>5</value>
  <description></description>
</property>

<property>
  <name>dfs.namenode.kerberos.internal.spnego.principal</name>
  <value>${dfs.web.authentication.kerberos.principal}</value>
</property>

<property>
  <name>dfs.secondary.namenode.kerberos.internal.spnego.principal</name>

  <value>${dfs.web.authentication.kerberos.principal}</value>
</property>

```

In addition, you must set the user on all secure DataNodes:

```

export HADOOP_SECURE_DN_USER=hdfs
export HADOOP_SECURE_DN_PID_DIR=/grid/0/var/run/hadoop/$HADOOP_SECURE_DN_USER

```

2.2.3.3. yarn-site.xml

You must add the following information to the `yarn-site.xml` file on every host in your cluster:

Table 2.7. yarn-site.xml

Property Name	Property Value	Description
<code>yarn.resourcemanager.principal</code>	<code>yarn/localhost@EXAMPLE.COM</code>	The Kerberos principal for the ResourceManager.
<code>yarn.resourcemanager.keytab</code>	<code>/etc/krb5.keytab</code>	The keytab for the ResourceManager.

Property Name	Property Value	Description
yarn.nodemanager.principal	yarn/localhost@EXAMPLE.COM	The Kerberos principal for the NodeManager.
yarn.nodemanager.keytab	/etc/krb5.keytab	The keytab for the NodeManager.
yarn.nodemanager.container-executor.class	org.apache.hadoop.yarn.server	The class that will execute (launch) the containers.
yarn.nodemanager.linux-container-executor.path	hadoop-3.0.0-SNAPSHOT/bin/container-executor	The path to the Linux container executor.
yarn.nodemanager.linux-container-executor.group	hadoop	A special group (e.g. hadoop) with executable permissions for the container executor, of which the NodeManager Unix user is the group member and no ordinary application user is. If any application user belongs to this special group, security will be compromised. This special group name should be specified for the configuration property.
yarn.timeline-service.principal	yarn/localhost@EXAMPLE.COM	The Kerberos principal for the Timeline Server.
yarn.timeline-service.keytab	/etc/krb5.keytab	The Kerberos keytab for the Timeline Server.
yarn.resourcemanager.webapp.delegation-token-auth-filter.enabled		Flag to enable override of the default Kerberos authentication filter with the RM authentication filter to allow authentication using delegation tokens (fallback to Kerberos if the tokens are missing). Only applicable when the http authentication type is Kerberos.
yarn.timeline-service.http-authentication.type	kerberos	Defines authentication used for the Timeline Server HTTP endpoint. Supported values are: simple kerberos \$AUTHENTICATION_HANDLER_CLASSNAME
yarn.timeline-service.http-authentication.kerberos.principal	HTTP/localhost@EXAMPLE.COM	The Kerberos principal to be used for the Timeline Server HTTP endpoint.
yarn.timeline-service.http-authentication.kerberos.keytab	/etc/krb5.keytab	The Kerberos keytab to be used for the Timeline Server HTTP endpoint.

The XML for these entries:

```
<property>
  <name>yarn.resourcemanager.principal</name>
  <value>yarn/localhost@EXAMPLE.COM</value>
</property>

<property>
  <name>yarn.resourcemanager.keytab</name>
  <value>/etc/krb5.keytab</value>
</property>

<property>
  <name>yarn.nodemanager.principal</name>
  <value>yarn/localhost@EXAMPLE.COM</value>
</property>

<property>
  <name>yarn.nodemanager.keytab</name>
  <value>/etc/krb5.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.path</name>
  <value>hadoop-3.0.0-SNAPSHOT/bin/container-executor</value>
</property>

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

<property>
  <name>yarn.timeline-service.principal</name>
  <value>yarn/localhost@EXAMPLE.COM</value>
</property>

<property>
  <name>yarn.timeline-service.keytab</name>
  <value>/etc/krb5.keytab</value>
</property>

<property>
  <name>yarn.resourcemanager.webapp.delegation-token-auth-filter.enabled</
name>
  <value>>true</value>
</property>

<property>
  <name>yarn.timeline-service.http-authentication.type</name>
  <value>kerberos</value>
</property>

<property>
  <name>yarn.timeline-service.http-authentication.kerberos.principal</name>
  <value>HTTP/localhost@EXAMPLE.COM</value>
</property>

<property>
  <name>yarn.timeline-service.http-authentication.kerberos.keytab</name>
  <value>/etc/krb5.keytab</value>
</property>

```

2.2.3.4. mapred-site.xml

To the `mapred-site.xml` file on every host in your cluster, you must add the following information:

Table 2.8. mapred-site.xml

Property Name	Property Value	Description	Final
<code>mapreduce.jobtracker.kerberos.principal</code>	<code>HTTP/localhost@EXAMPLE.COM</code>	Kerberos principal name for the JobTracker	

Property Name	Property Value	Description	Final
mapreduce.tasktracker.kerberos.principal	jt/_HOST@EXAMPLE.COM	Kerberos principal name for the TaskTracker. "_HOST" is replaced by the host name of the task tracker.	
hadoop.job.history.useonelocation	one		true
mapreduce.jobtracker.keytabsecurity	keytabs/ keytabs/ jt.service.keytab	The keytab for the JobTracker principal	
mapreduce.tasktracker.keytabsecurity	keytabs/ keytabs/ tt.service.keytab	The keytab for the Tasktracker principal	
mapreduce.jobtracker.staging.root.dir	staging.root.dir	The path prefix for the location of the staging directories. The next level is always the user's name. It is a path in the default file system	
mapreduce.tasktracker.group	hadoop	The group that the task controller uses for accessing the task controller. The mapred user must be a member and users should not be members. <question?>	
mapreduce.jobtracker.split.metainfo.maxsize	5000000	If the size of the split metainfo file is larger than this value, the JobTracker will fail the job during initialization.	true
mapreduce.history.server.embedded	embedded	Should the Job History server be embedded within the JobTracker process	true
mapreduce.history.server.example.address	Example Note: cluster variant ip-10-111-59-170.ec2.internal:51111		
mapreduce.jobhistory.kerberos.principal	jt/_HOST@EXAMPLE.COM	Kerberos principal name for JobHistory. This must map to the same user as the JT user.	true
mapreduce.jobhistory.keytabsecurity	keytabs/ keytabs/ jt.service.keytab	The keytab for the JobHistory principal	
mapred.jobtracker.blackboard.timeout.window	Example 180	3-hour sliding window - the value is specified in minutes.	
mapred.jobtracker.blackboard.bucket-width	Example 15	15-minute bucket size - the value is specified in minutes.	
mapred.queue.names	default	Comma separated list of queues configured for this JobTracker.	

The XML for these entries:

```
<property>
  <name>mapreduce.jobtracker.kerberos.principal</name>
  <value>jt/_HOST@EXAMPLE.COM</value>
  <description> JT
  user name key. </description>
```



```
</property>

<property>
  <name>mapreduce.tasktracker.kerberos.principal</name>
  <value>tt/_HOST@EXAMPLE.COM</value>
  <description>tt
  user name key. "_HOST" is replaced by the host name of the task
  tracker.
  </description>
</property>

<property>
  <name>hadoop.job.history.user.location</name>
  <value>none</value>
  <final>>true</final>
</property>

<property>
  <name>mapreduce.jobtracker.keytab.file</name>
  <value>/etc/security/keytabs/jt.service.keytab</value>
  <description>
  The keytab for the jobtracker principal.
  </description>
</property>

<property>
  <name>mapreduce.tasktracker.keytab.file</name>
  <value>/etc/security/keytabs/tt.service.keytab</value>
  <description>The filename of the keytab for the task
  tracker</description>
</property>

<property>
  <name>mapreduce.jobtracker.staging.root.dir</name>
  <value>/user</value>
  <description>The Path prefix for where the staging
  directories should be placed. The next level is always the user's
  name. It
  is a path in the default file system.</description>
</property>

<property>
  <name>mapreduce.tasktracker.group</name>
  <value>hadoop</value>
  <description>The group that the task controller uses for accessing the
  task controller.
  The mapred user must be a member and users should *not* be
  members.</description>
</property>

<property>
  <name>mapreduce.jobtracker.split.metainfo.maxsize</name>
  <value>50000000</value>
  <final>>true</final>
  <description>If the size of the split metainfo file is larger than
  this, the JobTracker
  will fail the job during
  initialize.
  </description>
</property>
```

```
<property>
  <name>mapreduce.history.server.embedded</name>
  <value>>false</value>
  <description>Should job history server be embedded within Job tracker
  process</description>
  <final>>true</final>
</property>

<property>
  <name>mapreduce.history.server.http.address</name>
  <!--cluster variant -->
  <value>ip-10-111-59-170.ec2.internal:51111</value>
  <description>Http address of the history server</description>
  <final>>true</final>
</property>

<property>
  <name>mapreduce.jobhistory.kerberos.principal</name>
  <!--cluster variant -->
  <value>jt/_HOST@EXAMPLE.COM</value>
  <description>Job history user name key. (must map to same user as JT
  user)</description>
</property>

<property>
  <name>mapreduce.jobhistory.keytab.file</name>
  <!--cluster variant -->
  <value>/etc/security/keytabs/jt.service.keytab</value>
  <description>The keytab for the job history server
  principal.</description>
</property>

<property>
  <name>mapred.jobtracker.blacklist.fault-timeout-window</name>
  <value>180</value>
  <description>      3-hour
  sliding window (value is in minutes)
  </description>
</property>

<property>
  <name>mapred.jobtracker.blacklist.fault-bucket-width</name>
  <value>15</value>
  <description>
  15-minute bucket size (value is in minutes)
  </description>
</property>

<property>
  <name>mapred.queue.names</name>
  <value>default</value>  <description>
  Comma separated list of queues configured for this jobtracker.</
  description>
</property>
```

2.2.3.5. hbase-site.xml

For HBase to run on a secured cluster, HBase must be able to authenticate itself to HDFS. To the `hbase-site.xml` file on your HBase server, you must add the following information. There are no default values; the following are all only examples:

Table 2.9. hbase-site.xml

Property Name	Property Value	Description
<code>hbase.master.keytab.file</code>	<code>/etc/security/keytabs/hm.service.keytab</code>	The keytab for the HMaster service principal
<code>hbase.master.kerberos.principal</code>	<code>hbase/_HOST@EXAMPLE.COM</code>	The Kerberos principal name that should be used to run the HMaster process. If <code>_HOST</code> is used as the hostname portion, it will be replaced with the actual hostname of the running instance.
<code>hbase.regionserver.keytab.file</code>	<code>/etc/security/keytabs/rs.service.keytab</code>	The keytab for the HRegionServer service principal
<code>hbase.regionserver.kerberos.principal</code>	<code>hbase/_HOST@EXAMPLE.COM</code>	The Kerberos principal name that should be used to run the HRegionServer process. If <code>_HOST</code> is used as the hostname portion, it will be replaced with the actual hostname of the running instance.
<code>hbase.superuser</code>	<code>hbase</code>	Comma-separated List of users or groups that are allowed full privileges, regardless of stored ACLs, across the cluster. Only used when HBase security is enabled.
<code>hbase.coprocessor.region.classes</code>		Comma-separated list of Coprocessors that are loaded by default on all tables. For any override coprocessor method, these classes will be called in order. After implementing your own Coprocessor, just put it in HBase's classpath and add the fully qualified class name here. A coprocessor can also be loaded on demand by setting <code>HTableDescriptor</code> .
<code>hbase.coprocessor.master.classes</code>		Comma-separated list of <code>org.apache.hadoop.hbase.coprocessor.MasterObserver</code> coprocessors that are loaded by default on the active HMaster process. For any implemented coprocessor methods, the listed classes will be called in order. After implementing your own <code>MasterObserver</code> , just put it in HBase's classpath and add the fully qualified class name here.

The XML for these entries:

```
<property>
  <name>hbase.master.keytab.file</name>
  <value>/etc/security/keytabs/hm.service.keytab</value>
  <description>Full path to the kerberos keytab file to use for logging
  in the configured HMaster server principal.
  </description>
</property>
```

```
<property>
  <name>hbase.master.kerberos.principal</name>
  <value>hm/_HOST@EXAMPLE.COM</value>
  <description>Ex. "hbase/_HOST@EXAMPLE.COM".
  The kerberos principal name that
  should be used to run the HMaster process. The
  principal name should be in
  the form: user/hostname@DOMAIN. If "_HOST" is used
  as the hostname portion, it will be replaced with the actual hostname
  of the running
  instance.
  </description>
</property>

<property>
  <name>hbase.regionserver.keytab.file</name>
  <value>/etc/security/keytabs/rs.service.keytab</value>
  <description>Full path to the kerberos keytab file to use for logging
  in the configured HRegionServer server principal.
  </description>
</property>

<property>
  <name>hbase.regionserver.kerberos.principal</name>
  <value>rs/_HOST@EXAMPLE.COM</value>
  <description>Ex. "hbase/_HOST@EXAMPLE.COM".
  The kerberos principal name that
  should be used to run the HRegionServer process. The
  principal name should be in the form:
  user/hostname@DOMAIN. If _HOST
  is used as the hostname portion, it will be replaced
  with the actual hostname of the running
  instance. An entry for this principal must exist
  in the file specified in hbase.regionserver.keytab.file
  </description>
</property>

<!--Additional configuration specific to HBase security -->

<property>
  <name>hbase.superuser</name>
  <value>hbase</value>
  <description>List of users or groups (comma-separated), who are
  allowed full privileges, regardless of stored ACLs, across the
  cluster. Only
  used when HBase security is enabled.
  </description>
</property>

<property>
  <name>hbase.coprocessor.region.classes</name>
  <value></value>
  <description>A comma-separated list of Coprocessors that are loaded
  by default on all tables. For any override coprocessor method, these
  classes
  will be called in order. After implementing your own Coprocessor,
  just put it in HBase's classpath and add the fully qualified class
  name here. A
  coprocessor can also be loaded on demand by setting HTableDescriptor.
```

```

    </description>
  </property>

  <property>
    <name>hbase.coprocessor.master.classes</name>
    <value></value>
    <description>A comma-separated list of
    org.apache.hadoop.hbase.coprocessor.MasterObserver coprocessors that
    are loaded by default on the active HMaster process. For any
    implemented
    coprocessor methods, the listed classes will be called in order.
    After implementing your own MasterObserver, just put it in HBase's
    classpath and add the fully qualified class name here.
    </description>
  </property>

```

2.2.3.6. hive-site.xml

Hive Metastore supports Kerberos authentication for Thrift clients only. HiveServer does not support Kerberos authentication for any clients:

Table 2.10. hive-site.xml

Property Name	Property Value	Description
hive.metastore.sasl.enabled	true	If true, the Metastore Thrift interface will be secured with SASL and clients must authenticate with Kerberos
hive.metastore.kerberos.keytab.file	/etc/security/keytabs/hive.service.keytab	The keytab for the Metastore Thrift service principal
hive.metastore.kerberos.principal	hive/_HOST@EXAMPLE.COM	The service principal for the Metastore Thrift server. If _HOST is used as the hostname portion, it will be replaced with the actual hostname of the running instance.
hive.metastore.cache.pinobjtypes	Table, Database, Type, FieldSchema, Order	Comma-separated Metastore object types that should be pinned in the cache

The XML for these entries:

```

<property>
  <name>hive.metastore.sasl.enabled</name>
  <value>true</value>
  <description>If true, the metastore thrift interface will be secured
  with
  SASL.
  Clients must authenticate with Kerberos.</description>
</property>

<property>
  <name>hive.metastore.kerberos.keytab.file</name>
  <value>/etc/security/keytabs/hive.service.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/_HOST@EXAMPLE.COM</value>
<description>The service principal for the metastore thrift server.
The
special string _HOST will be replaced automatically with the correct
hostname.</description>
</property>
<property>
<name>hive.metastore.cache.pinobjtypes</name>
<value>Table,Database,Type,FieldSchema,Order</value>
<description>List of comma separated metastore object types that
should be pinned in
the cache</description>
</property>

```

2.2.3.6.1. oozie-site.xml

To the oozie-site.xml file, you must add the following information:

Table 2.11. oozie-site.xml

Property Name	Property Value	Description
oozie.service.AuthorizationService.security.enabled	Service.security.enabled	Specifies whether security (user name/admin role) is enabled or not. If it is disabled any user can manage the Oozie system and manage any job.
oozie.service.HadoopAccessorService.kerberos.enabled	Service.kerberos.enabled	Indicates if Oozie is configured to use Kerberos
local.realm	EXAMPLE.COM	Kerberos Realm used by Oozie and Hadoop. Using local.realm to be aligned with Hadoop configuration.
oozie.service.HadoopAccessorService.security/keytabs/oozie.service.keytab	Service/security/keytabs/oozie.service.keytab	The keytab for the Oozie service principal.
oozie.service.HadoopAccessorService_principal	Service_principal@EXAMPLE.COM	Kerberos principal for Oozie service
oozie.authentication.type	kerberos	
oozie.authentication.kerberos_principal	Service_principal@EXAMPLE.COM	Whitelisted job tracker for Oozie service
oozie.authentication.kerberos_keytab	Service/security/keytabs/spnego.service.keytab	Location of the Oozie user keytab file.
oozie.service.HadoopAccessorService.nameNode.whitelist	Service.nameNode.whitelist	
oozie.authentication.kerberos_principal_to_localhost_mapping	RULE:[2:\$1@\$0]([jt]t@.*EXAMPLE.COM)s/.*mapred/ RULE:[2:\$1@\$0]([nd]n@.*EXAMPLE.COM)s/.*hdfs/ RULE:[2:\$1@\$0](hm@.*EXAMPLE.COM)s/.*hbase/ RULE:[2:\$1@\$0](rs@.*EXAMPLE.COM)s/.*hbase/ DEFAULT	The mapping from Kerberos principal names to local OS user names. See Creating Mappings Between Principals and UNIX Usernames for more information.
oozie.service.ProxyUserService.proxyuser.knox.groups	proxyuser.knox.groups	Grant proxy privileges to the Knox user. Note only required when using a Knox Gateway.
oozie.service.ProxyUserService.proxyuser.knox.hosts	\$knoxuser_FQDN.knox.hosts	Identifies the Knox Gateway. Note only required when using a Knox Gateway.

2.2.3.6.2. webhcat-site.xml

To the webhcat-site.xml file, you must add the following information:

Table 2.12. webhcat-site.xml

Property Name	Property Value	Description
templeton.kerberos.principal	HTTP/_HOST@EXAMPLE.COM	
templeton.kerberos.keytab	abc/security/keytabs/spnego.service.keytab	
templeton.kerberos.secret	secret	
hadoop.proxyuser.knox.groups	groups	Grant proxy privileges to the Knox user. Note only required when using a Knox Gateway.
hadoop.proxyuser.knox.knox_host_FQDN	hostname	Identifies the Knox Gateway. Note only required when using a Knox Gateway.

2.3. Configure Secure HBase and ZooKeeper

Use the following instructions to set up secure HBase and ZooKeeper:

1. [Configure HBase Master](#)
2. [Create JAAS configuration files](#)
3. [Start HBase and ZooKeeper services](#)
4. [Configure secure client side access for HBase](#)
5. [Optional: Configure client-side operation for secure operation - Thrift Gateway](#)
6. [Optional: Configure client-side operation for secure operation - REST Gateway](#)
7. [Configure HBase for Access Control Lists \(ACL\)](#)

2.3.1. Configure HBase Master

Edit `$HBASE_CONF_DIR/hbase-site.xml` file on your HBase Master server to add the following information (`$HBASE_CONF_DIR` is the directory to store the HBase configuration files. For example, `/etc/hbase/conf`):



Note

There are no default values. The following are all examples.

```
<property>
  <name>hbase.master.keytab.file</name>
  <value>/etc/security/keytabs/hbase.service.keytab</value>
  <description>Full path to the kerberos keytab file to use
    for logging in the configured HMaster server principal.

  </description>
</property>
```

```
<property>
  <name>hbase.master.kerberos.principal</name>
  <value>hbase/_HOST@EXAMPLE.COM</value>
  <description>Ex. "hbase/_HOST@EXAMPLE.COM".
    The kerberos principal name that should be used to run the HMaster
    process.
    The principal name should be in the form: user/hostname@DOMAIN. If
    "_HOST" is used as the hostname portion,
    it will be replaced with the actual hostname of the running instance.

  </description>
</property>
```

```
<property>
  <name>hbase.regionserver.keytab.file</name>
  <value>/etc/security/keytabs/hbase.service.keytab</value>
  <description>Full path to the kerberos keytab file to use for logging
    in the configured HRegionServer server principal.

  </description>
</property>
```

```
<property>
  <name>hbase.regionserver.kerberos.principal</name>
  <value>hbase/_HOST@EXAMPLE.COM</value>
  <description>Ex. "hbase/_HOST@EXAMPLE.COM".The kerberos principal name
    that should be used to run the HRegionServer process.
    The principal name should be in the form: user/hostname@DOMAIN.
    If _HOST is used as the hostname portion, it will be replaced with the actual
    hostname of the running instance.
    An entry for this principal must exist in the file specified in hbase.
    regionserver.keytab.file

  </description>
</property>
```

```
<!--Additional configuration specific to HBase security -->
```

```
<property>
  <name>hbase.superuser</name>
  <value>hbase</value>
  <description>List of users or groups (comma-separated), who are
    allowed full privileges, regardless of stored ACLs, across the cluster.
    Only used when HBase security is enabled.

  </description>
</property>
```



```
<property>
  <name>hbase.coprocessor.region.classes</name>
  <value>org.apache.hadoop.hbase.security.token.TokenProvider,org.
apache.hadoop.hbase.security.access.SecureBulkLoadEndpoint,org.apache.hadoop.
hbase.security.access.AccessController </value>
  <description>A comma-separated list of Coprocessors that are loaded by
default on all tables.
</description>
</property>
```

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

```
<property>
  <name>hbase.rpc.engine</name>
  <value>org.apache.hadoop.hbase.ipc.SecureRpcEngine</value>
</property>
```

```
<property>
  <name>hbase.security.authorization</name>
  <value>true</value>
  <description>Enables HBase authorization. Set the value of this
property to false to disable HBase authorization.
</description>
</property>
```

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

```
<property>
  <name>hbase.bulkload.staging.dir</name>
  <value>/apps/hbase/staging</value>
  <description>Directory in the default filesystem, owned by the hbase
user, and has permissions(-rwx--x--x, 711) </description>
</property>
```

For more information on bulk loading in secure mode, see [HBase Secure BulkLoad](#). Note that the `hbase.bulkload.staging.dir` is created by HBase.

2.3.2. Create JAAS configuration files

1. Create the following JAAS configuration files on the HBase Master, RegionServer, and HBase client host machines.

These files must be created under the `$HBASE_CONF_DIR` directory:

where `$HBASE_CONF_DIR` is the directory to store the HBase configuration files. For example, `/etc/hbase/conf`.

- On your HBase Master host machine, create the `hbase-server.jaas` file under the `/etc/hbase/conf` directory and add the following content:

```
Server {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  useTicketCache=false
  keyTab="/etc/security/keytabs/hbase.service.keytab"
  principal="hbase/$HBase.Master.hostname";
};
```

- On each of your RegionServer host machine, create the `regionserver.jaas` file under the `/etc/hbase/conf` directory and add the following content:

```
Server {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  useTicketCache=false
  keyTab="/etc/security/keytabs/hbase.service.keytab"
  principal="hbase/$RegionServer.hostname";
};
```

- On HBase client machines, create the `hbase-client.jaas` file under the `/etc/hbase/conf` directory and add the following content:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=false
  useTicketCache=true;
};
```

2. Create the following JAAS configuration files on the ZooKeeper Server and client host machines.

These files must be created under the `$ZOOKEEPER_CONF_DIR` directory, where `$ZOOKEEPER_CONF_DIR` is the directory to store the HBase configuration files. For example, `/etc/zookeeper/conf`:

- On ZooKeeper server host machines, create the `zookeeper-server.jaas` file under the `/etc/zookeeper/conf` directory and add the following content:

```
Server {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  useTicketCache=false
  keyTab="/etc/security/keytabs/zookeeper.service.keytab"
  principal="zookeeper/$ZooKeeper.Server.hostname";
};
```

- On ZooKeeper client host machines, create the `zookeeper-client.jaas` file under the `/etc/zookeeper/conf` directory and add the following content:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=false
  useTicketCache=true;
};
```

3. Edit the `hbase-env.sh` file on your HBase server to add the following information:

```
export HBASE_OPTS="-Djava.security.auth.login.config=$HBASE_CONF_DIR/hbase-client.jaas"
export HBASE_MASTER_OPTS="-Djava.security.auth.login.config=$HBASE_CONF_DIR/hbase-server.jaas"
export HBASE_REGIONSERVER_OPTS="-Djava.security.auth.login.config=$HBASE_CONF_DIR/regionserver.jaas"
```

where `HBASE_CONF_DIR` is the HBase configuration directory. For example, `/etc/hbase/conf`.

4. Edit `zoo.cfg` file on your ZooKeeper server to add the following information:

```
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
jaasLoginRenew=3600000
kerberos.removeHostFromPrincipal=true
kerberos.removeRealmFromPrincipal=true
```

5. Edit `zookeeper-env.sh` file on your ZooKeeper server to add the following information:

```
export SERVER_JVMFLAGS="-Djava.security.auth.login.config=$ZOOKEEPER_CONF_DIR/zookeeper-server.jaas"
export CLIENT_JVMFLAGS="-Djava.security.auth.login.config=$ZOOKEEPER_CONF_DIR/zookeeper-client.jaas"
```

where `$ZOOKEEPER_CONF_DIR` is the ZooKeeper configuration directory. For example, `/etc/zookeeper/conf`.

2.3.3. Start HBase and ZooKeeper services

Start the HBase and ZooKeeper services using the instructions provided [here](#).

If the configuration is successful, you should see the following in your ZooKeeper server logs:

```
11/12/05 22:43:39 INFO zookeeper.Login: successfully logged in.
11/12/05 22:43:39 INFO server.NIOServerCnxnFactory: binding to port 0.0.0.0/0.0.0.0:2181
11/12/05 22:43:39 INFO zookeeper.Login: TGT refresh thread started.
11/12/05 22:43:39 INFO zookeeper.Login: TGT valid starting at:          Mon Dec
05 22:43:39 UTC 2011
11/12/05 22:43:39 INFO zookeeper.Login: TGT expires:                  Tue Dec
06 22:43:39 UTC 2011
11/12/05 22:43:39 INFO zookeeper.Login: TGT refresh sleeping until: Tue Dec 06
18:36:42 UTC 2011
..
11/12/05 22:43:59 INFO auth.SaslServerCallbackHandler:
  Successfully authenticated client: authenticationID=hbase/ip-10-166-175-249.
us-west-1.compute.internal@HADOOP.LOCALDOMAIN;
  authorizationID=hbase/ip-10-166-175-249.us-west-1.compute.internal@HADOOP.
LOCALDOMAIN.
11/12/05 22:43:59 INFO auth.SaslServerCallbackHandler: Setting authorizedID:
  hbase
11/12/05 22:43:59 INFO server.ZooKeeperServer: adding SASL authorization for
  authorizationID: hbase
```

2.3.4. Configure secure client side access for HBase

HBase configured for secure client access is expected to be running on top of a secure HDFS cluster. HBase must be able to authenticate to HDFS services.

1. Provide a Kerberos principal to the HBase client user using the instructions provided [here](#).

- **Option I:** Provide Kerberos principal to normal HBase clients.

For normal HBase clients, Hortonworks recommends setting up a password to the principal.

- Set `maxrenewlife`.

The client principal's `maxrenewlife` should be set high enough so that it allows enough time for the HBase client process to complete. Client principals are not renewed automatically.

For example, if a user runs a long-running HBase client process that takes at most three days, we might create this user's principal within `kadmin` with the following command:

```
addprinc -maxrenewlife 3days
```

- **Option II:** Provide Kerberos principal to long running HBase clients.
 - a. Set-up a keytab file for the principal and copy the resulting keytab files to where the client daemon will execute.

Ensure that you make this file readable only to the user account under which the daemon will run.

2. On every HBase client, add the following properties to the `$HBASE_CONF_DIR/hbase-site.xml` file:

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



Note

The client environment must be logged in to Kerberos from KDC or keytab via the `kinit` command before communication with the HBase cluster is possible. Note that the client will not be able to communicate with the cluster if the `hbase.security.authentication` property in the client- and server-side site files fails to match.

```
<property>
  <name>hbase.rpc.engine</name>
  <value>org.apache.hadoop.hbase.ipc.SecureRpcEngine</value>
</property>
```

2.3.5. Optional: Configure client-side operation for secure operation - Thrift Gateway

Add the following to the `$HBASE_CONF_DIR/hbase-site.xml` file for every Thrift gateway:

```
<property>
  <name>hbase.thrift.keytab.file</name>
  <value>/etc/hbase/conf/hbase.keytab</value>
</property>
<property>
  <name>hbase.thrift.kerberos.principal</name>
  <value>${USER}/_HOST@HADOOP.LOCALDOMAIN</value>
</property>
```

Substitute the appropriate credential and keytab for `$USER` and `$KEYTAB` respectively.

The Thrift gateway will authenticate with HBase using the supplied credential. No authentication will be performed by the Thrift gateway itself. All client access via the Thrift gateway will use the Thrift gateway's credential and have its privilege.

2.3.6. Optional: Configure client-side operation for secure operation - REST Gateway

Add the following to the `$HBASE_CONF_DIR/hbase-site.xml` file for every REST gateway:

```
<property>
  <name>hbase.rest.keytab.file</name>
  <value>${KEYTAB}</value>
</property>
<property>
  <name>hbase.rest.kerberos.principal</name>
  <value>${USER}/_HOST@HADOOP.LOCALDOMAIN</value>
</property>
```

Substitute the appropriate credential and keytab for *USER* and *KEYTAB* respectively.

The REST gateway will authenticate with HBase using the supplied credential. No authentication will be performed by the REST gateway itself. All client access via the REST gateway will use the REST gateway's credential and have its privilege.

2.3.7. Configure HBase for Access Control Lists (ACL)

Use the following instructions to configure HBase for ACL:

1. Open `kinit` as HBase user.
 - a. Create a keytab for principal `hbase@REALM` and store it in the `hbase.headless.keytab` file. See instructions provided [here](#) for creating principal and keytab file.
 - b. Open `kinit` as HBase user. Execute the following command on your HBase Master:

```
kinit -kt hbase.headless.keytab hbase
```

2. Start the HBase shell. On the HBase Master host machine, execute the following command:

```
hbase shell
```

3. Set ACLs using HBase shell:

```
grant '$USER', '$permissions'
```

where

- *USER* is any user responsible for create/update/delete operations in HBase.



Note

You must set the ACLs for all those users who will be responsible for create/update/delete operations in HBase.

- *permissions* is zero or more letters from the set "RWCA": READ('R'), WRITE('W'), CREATE('C'), ADMIN('A').

2.4. Setting up One-Way Trust with Active Directory

In environments where users from Active Directory (AD) need to access Hadoop Services, set up one-way trust between Hadoop Kerberos realm and the AD (Active Directory) domain.



Warning

Hortonworks recommends setting up one-way trust after fully configuring and testing your Kerberized Hadoop Cluster.

2.4.1. Configure Kerberos Hadoop Realm on the AD DC

Configure the Hadoop realm on the AD DC server and set up the one-way trust.

1. Add the Hadoop Kerberos realm and KDC host to the DC:

```
ksetup /addkdc $hadoop.realm $KDC-host
```

2. Establish one-way trust between the AD domain and the Hadoop realm:

```
netdom trust $hadoop.realm /Domain:$AD.domain /add /realm /  
passwordt:$trust_password
```

3. (Optional) If Windows clients within the AD domain need to access Hadoop Services, and the domain does not have a search route to find the services in Hadoop realm, run the following command to create a hostmap for Hadoop service host:

```
ksetup /addhosttorealmmap $hadoop-service-host $hadoop.realm
```



Note

Run the above for each *\$hadoop-host* that provides services that need to be accessed by Windows clients. For example, Oozie host, WebHCat host, etc.

4. (Optional) define the encryption type:

```
ksetup /SetEncTypeAttr $hadoop.realm $encryption_type
```

Set encryption types based on your security requirements. Mismatching encryption types causes problems.



Note

Run `ksetup /GetEncTypeAttr $krb_realm` to list the available encryption types. Verify that the encryption type is configured for the Hadoop realm in the `krb5.conf`.

2.4.2. Configure the AD Domain on the KDC and Hadoop Cluster Hosts

Add the AD domain as a realm to the `krb5.conf` on the Hadoop cluster hosts. Optionally configure encryption types and UDP preferences.

1. Open the `krb5.conf` file with a text editor and make the following changes:

a. To `libdefaults`, add the following properties:

Sets the Hadoop realm as default:

```
[libdefaults]
default_domain = $hadoop.realm
```

Set the encryption type:

```
[libdefaults]
default_tkt_enctypes = $encryption_types
default_tgs_enctypes = $encryption_types
permitted_enctypes = $encryption_types
```

where the `$encryption_types` match the type supported by your environment.

For example:

```
default_tkt_enctypes = aes256-cts aes128-cts rc4-hmac arcfour-hmac-md5
des-cbc-md5 des-cbc-crc
default_tgs_enctypes = aes256-cts aes128-cts rc4-hmac arcfour-hmac-md5
des-cbc-md5 des-cbc-crc
permitted_enctypes = aes256-cts aes128-cts rc4-hmac arcfour-hmac-md5 des-
cbc-md5 des-cbc-crc
```

If TCP is open on the KDC and AD Server:

```
[libdefaults]
udp_preference_limit = 1
```

b. Add a realm for the AD domain:

```
[realms]
$AD.DOMAIN = {
  kdc = $AD-host-FQDN
  admin_server = $AD-host-FQDN
  default_domain = $AD-host-FQDN
}
```

c. Save the `krb5.conf` the changes to all Hadoop Cluster hosts.

2. Add the trust principal for the AD domain to the Hadoop MIT KDC:

```
kadmin
kadmin: addprinc krbtgt/$hadoop.realm@$AD.domain
```

This command will prompt you for the trust password, use the same password as the earlier step.



Note

If the encryption type was defined, then use the following command to configure the AD principal:

```
kadmin: addprinc -e "$encryption_type" krbtgt/$hadoop.  
realm@$AD.domain
```

2.5. Allowing Impersonation

<http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/Supersusers.html>

3. Data Protection

Encryption is applied to electronic information in order to ensure its privacy and confidentiality. Wire encryption protects data as it moves into and through Hadoop cluster over RPC, HTTP, Data Transfer Protocol (DTP), and JDBC.

The following describes how the data is protected as it is in motion:

- Clients typically communicate directly with the Hadoop cluster and the data can be protected using:
 - **RPC encryption:** Clients interacting directly with the Hadoop cluster through RPC. A client uses RPC to connect to the NameNode (NN) to initiate file read and write operations. RPC connections in Hadoop use Java's Simple Authentication & Security Layer (SASL), which supports encryption.
 - **Data Transfer Protocol:** The NN gives the client the address of the first DataNode (DN) to read or write the block. The actual data transfer between the client and a DN uses Data Transfer Protocol.
- Users typically communicate with the Hadoop cluster using a Browser or a command line tools, data can be protected using:
 - **HTTPS encryption:** Users typically interact with Hadoop using a browser or component CLI, while applications use REST APIs or Thrift. Encryption over the HTTP protocol is implemented with the support for SSL across a Hadoop cluster and for the individual components such as Ambari.
 - **JDBC:** HiveServer2 implements encryption with Java SASL protocol's quality of protection (QOP) setting. With this the data moving between a HiveServer2 over jdbc and a jdbc client can be encrypted.
- Additionally within the cluster communication between processes can be protected using:
 - **HTTPS encryption during shuffle:** Starting in HDP 2.0 encryption during shuffle is supported. The data moves between the Mappers and the Reducers over the HTTP protocol, this step is called shuffle. Reducer initiates the connection to the Mapper to ask for data and acts as SSL client.

This chapter provides an overview on encryption over-the-wire in Hadoop. Data can be moved in and out of Hadoop over RPC, HTTP, Data Transfer Protocol, and JDBC. Network traffic over each of these protocols can be encrypted to provide privacy for data movement.

- [Section 3.1, "Enable RPC Encryption for the Hadoop Cluster" \[39\]](#)
- [Section 3.2, "Enable Data Transfer Protocol" \[39\]](#)
- [Section 3.3, "Enable SSL on HDP Components" \[39\]](#)
- [Section 3.4, "Connect to SSL Enabled Components" \[55\]](#)

3.1. Enable RPC Encryption for the Hadoop Cluster

The most common way for a client to interact with a Hadoop cluster is through RPC. A client connects to a NameNode over RPC protocol to read or write a file. RPC connections in Hadoop use the Java Simple Authentication and Security Layer (SASL) which supports encryption. When the `hadoop.rpc.protection` property is set to `privacy`, the data over RPC is encrypted with symmetric keys.



Note

RPC encryption covers not only the channel between a client and a Hadoop cluster but also the inter-cluster communication among Hadoop services.

Enable Encrypted RPC by setting the following properties in `core-site.xml`.

```
hadoop.rpc.protection=privacy
```

(Also supported are the 'authentication' and 'integrity' settings.)

3.2. Enable Data Transfer Protocol

The NameNode gives the client the address of the first DataNode to read or write the block. The actual data transfer between the client and the DataNode is over Hadoop's Data Transfer Protocol. To encrypt this protocol you must set `dfs.encrypt.data.transfer=true` on the NameNode and all DataNodes. The actual algorithm used for encryption can be customized with `dfs.encrypt.data.transfer.algorithm` set to either "3des" or "rc4". If nothing is set, then the default on the system is used (usually 3DES.) While 3DES is more cryptographically secure, RC4 is substantially faster.

Enable Encrypted DTP by setting the following properties in `hdfs-site.xml`:

```
dfs.encrypt.data.transfer=true
dfs.encrypt.data.transfer.algorithm=3des
```

('rc4' is also supported.)



Note

Secondary Namenode is not supported with the HTTPS port. It can only be accessed by "http://<SNN>:50090". WebHDFS, hdfs, and shortcircuitread are not supported with SSL enabled.

3.3. Enable SSL on HDP Components

The following table links to step-by-step instructions to enable SSL in the HDP environment:

Table 3.1. Configure Data Protection for HDP Components

HDP Component	Notes/Link
Hadoop	Section 3.1, "Enable RPC Encryption for the Hadoop Cluster" [39] Section 3.3.3, "Enable SSL for WebHDFS, MapReduce Shuffle, and YARN" [49]

HDP Component	Notes/Link
HBase	Section 3.3.5, "Enable SSL on WebHBase and the HBase REST API" [53]
Ambari Server	Ambari Security Guide, Advanced Security Options, Optional: Set Up Security for Ambari
Oozie	Section 3.3.4, "Enable SSL on Oozie" [52]
Falcon	Enabled by default, to disable see FALCON-229
Sqoop	Client of Hive and HBase, see Data Integration Services with HDP, Apache Sqoop Connectors
Knox Gateway	Apache Knox Gateway Administrator Guide, Gateway Security, Configure Wire Encryption
Flume	Apache Flume User Guide, Defining Sources
Accumulo	Apache Foundation Blog, Apache Accumulo: Generating Keystores for configuring Accumulo with SSL
Phoenix	Installing HDP Manually, Installing Phoenix: Configuring Phoenix for Security and Apache Phoenix, Flume Plug-in
Hive (HiveServer2)	Section 3.3.6, "Enable SSL on HiveServer2" [54]
YARN	Section 3.3.3, "Enable SSL for WebHDFS, MapReduce Shuffle, and YARN" [49]
HUE	Installing HDP Manually, Installing Hue, (Optional.) Configure Hue for SSL
Ganglia	Ambari Security Guide, Advanced Security Options, Optional: Set Up Security for Ambari
Nagios	Ambari Security Guide, Advanced Security Options, Optional: Set Up Security for Ambari

This section contains the following topics:

- [Section 3.3.1, "Understanding Hadoop SSL Keystore Factory" \[40\]](#)
- [Section 3.3.2, "Manage SSL Certificates" \[42\]](#)
- [Section 3.3.3, "Enable SSL for WebHDFS, MapReduce Shuffle, and YARN" \[49\]](#)
- [Section 3.3.4, "Enable SSL on Oozie" \[52\]](#)
- [Section 3.3.5, "Enable SSL on WebHBase and the HBase REST API" \[53\]](#)
- [Section 3.3.6, "Enable SSL on HiveServer2" \[54\]](#)

3.3.1. Understanding Hadoop SSL Keystore Factory

The *Hadoop SSL Keystore Factory* manages SSL for Hadoop core services that communicate with other cluster services over HTTP, that is MapReduce, YARN, and HDFS. While components that have services that are typically not distributed or only receive HTTP connections directly from clients use the built-in Java JDK SSL tools, such as HBase, and Oozie.

The following table shows the Hadoop cluster services that use HTTP and support SSL for wire encryption.

Table 3.2. Components that Support SSL

Component	Service	SSL Management
HDFS	WebHDFS	Hadoop SSL Keystore Factory

Component	Service	SSL Management
MapReduce	Shuffle	Hadoop SSL Keystore Factory
	TaskTracker	Hadoop SSL Keystore Factory
Yarn	Resource Manager	Hadoop SSL Keystore Factory
	JobHistory	Hadoop SSL Keystore Factory
Hive	HiveServer2	Configured in hive-site.xml
HBase	REST API	Configured in hbase-site.xml
Solr		JDK: User and default
Accumulo		JDK: User and default
Falcon	REST API	JDK: User and default
Oozie		Configured in oozie-site.xml
Knox	Hadoop cluster (REST client)	JDK: default only
	Knox Gateway server	JDK: User and default
HDP Security Administration	Server/Agent	JDK: User and default

When enabling support for SSL, it is important to know which SSL Management method is used by the Hadoop service. Services that are co-located on a host must have the server certificate and keys, and in some cases the client truststore, configured in both the Hadoop SSL Keystore Factory and JDK locations. When using CA signed certificates, configure the Hadoop SSL Keystore Factory to use the Java keystore and truststore locations.

The following describes the major differences between certificates managed by JDK and the Hadoop SSL Keystore Factory:

- Hadoop SSL Keystore Management Factory:
 - Supports only JKS formatted keys
 - Supports toggling the shuffle between HTTP and HTTPS.
 - Supports two way certificate and name validation.
 - Uses a common location for both the keystore and truststore that is available to other Hadoop core services.
 - Allows you to manage SSL in a central location and propagate changes to all cluster nodes.
 - Automatically reloads the keystore and truststore without restarting services.
- SSL Management with JDK:
 - Allows either HTTP or HTTPS.
 - Uses hardcoded locations for truststores and keystores that may vary between hosts. And generally requires you to generate key pairs and import certificates on each host.
 - Requires the service to be restarted to reload the keystores and truststores.
 - Requires certificates to be installed in the client CA truststore.



Note

For more information on JDK SSL Management, see [Using SSL in Chapter 2: Monitoring and Managing Using JDK](#).

3.3.2. Manage SSL Certificates

When setting up SSL in production environments, Hortonworks recommends using Certificate Authority (CA) certificates that are signed by a well-known, trusted third-party. In development and other non-production environments, you may want to use certificates signed by an internal CA or self-signed certificate.

Java-based Hadoop components such as HDFS, MapReduce, YARN, etc support `JKS` format, while Python based services such as Hue uses `PEM` format.

The basic process for enabling SSL support includes:

- Create a signed certificate as follows:
 - From a third-party CA (recommended for production environments)
 - Creating and signing the certificate using an internal CA
 - Creating a self-signed certificate (not recommended, use only in small cluster for demonstration)
- Installing the certificate on the host
- Generating a public/private key pair
- Securing the private key
- Importing the certificate into the client truststore
- Enabling SSL for Hadoop Component
- Restarting the services

This section contains the following topics:

- [Section 3.3.2.1, "Generate a Certificate Signing Request \(Third-party\)" \[43\]](#)
- [Section 3.3.2.2, "Sign with Internal CA \(OpenSSL\)" \[45\]](#)
- [Section 3.3.2.3, "Create and Import Self-Signed Certificate" \[46\]](#)
- [Section 3.3.2.4, "Use a CA Signed Certificate" \[47\]](#)
- [Section 3.3.1, "Understanding Hadoop SSL Keystore Factory" \[40\]](#)



Note

These instructions are based on information found in the Oracle keytool reference, see [keytool - Key and Certificate Management Tool](#) and the OpenSSL [How to Documentation](#), refer to the SSL utilities for details.

3.3.2.1. Generate a Certificate Signing Request (Third-party)

To obtain a certificate signed by a third-party CA, generate a certificate signing request for each Hadoop Cluster host node as follows:



Note

A certificate signing request is typically not required when signing the certificate using an internal CA.

1. Using the service user account (shown below as `$service_user`), such as `hive`, `hbase`, `oozie`, `hdfs`, etc, generate the host key :

```
su -l $service_user -C "keytool -keystore keystore.jks -genkey -alias $host"

Enter keystore password: javacaps
What is your first and last name?
[Unknown]: name
What is the name of your organizational unit?
[Unknown]: department
What is the name of your organization?
[Unknown]: companyname
What is the name of your City or Locality?
[Unknown]: city
What is the name of your State or Province?
[Unknown]: state
What is the two-letter country code for this unit?
[Unknown]: XX
Is <CN=name, OU=department, O=companyname, L=city, ST=state, C=XX> correct?
[no]: yes

Enter key password for <host>
(RETURn if same as keystore password):
```

The keystore file is created in the user's home directory. Access to the keystore requires the password and alias.

2. Verify that key was generated:

```
su -l hdfs -c "keytool -list -v -keystore keystore.jks"
```

3. Create the Certificate Signing Request (CSR).

```
su -l hdfs -c "keytool -keystore $keystorename -certreq -alias $host -keyalg
rsa -file $host.csr"
```

The CSR is created in the user's home directory.

4. Verify that the file was created:

```
su -l hdfs -c "ls ~/"
cache keystore.jks $host.csr
```

5. Send the CSR to the CA. Follow the instructions sent to you by the CA to properly install the keys and certificate.



Note

Request generation instructions may vary depending on the third-party CAs, follow the instructions provided by them.

Example for HDFS user creating a signing request on host nn:

```
su -l hdfs -c "keytool -keystore keystore -genkey -alias nn"
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]: Hortonworks
What is the name of your organizational unit?
  [Unknown]: Engineering
What is the name of your organization?
  [Unknown]: Hortonworks
What is the name of your City or Locality?
  [Unknown]: Palo Alto
What is the name of your State or Province?
  [Unknown]: California
What is the two-letter country code for this unit?
  [Unknown]: US
Is CN=Hortonworks, OU=Engineering, O=Hortonworks, L=Palo Alto, ST=California,
C=US correct?
  [no]: yes

Enter key password for <nn>
(RETURN if same as keystore password):

su -l hdfs -c "keytool -list -v -keystore keystore.jks"
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: nn
Creation date: Aug 25, 2014
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Hortonworks, OU=Engineering, O=Hortonworks, L=Palo Alto, ST=
California, C=US
Issuer: CN=Hortonworks, OU=Engineering, O=Hortonworks, L=Palo Alto, ST=
California, C=US
Serial number: 53fb6404
Valid from: Mon Aug 25 16:27:48 UTC 2014 until: Sun Nov 23 16:27:48 UTC 2014
Certificate fingerprints:
  MD5:  58:97:7B:B4:47:0D:2B:97:3D:0D:AA:82:D6:1D:A3:38
  SHA1: 62:9F:EB:7F:DA:3A:F6:43:58:BF:43:A0:F2:BF:E0:16:A8:B2:D5:A9
  Signature algorithm name: SHA1withDSA
  Version: 3

*****
```



```
su -l hdfs -c "keytool -keystore keystore.jks -certreq -alias nn -keyalg rsa -
file nn.csr"
Enter keystore password:

su -l hdfs -c "ls ~/"
cache keystore.jks nn.csr
```

3.3.2.2. Sign with Internal CA (OpenSSL)

OpenSSL allows you sign certificates. Considerations for internally signed certificates include:

- The encryption algorithms may be less secure than a well-known, trusted third-party
- Unknown CAs require that certificate to be installed in the corresponding client truststores



Note

When accessing the service from a client application such as HiveCLI, cURL, etc, the CA must resolve on the client side or the connection attempt may fail. Users accessing the service through a browser will be able to add an exception if the certificate cannot be verified in their local truststore.

To install OpenSSL and set up an internal CA:

1. Install OpenSSL, for example on CentOS run:

```
yum install openssl
```

2. Generate a CA signing key and certificate:

```
openssl genrsa -out ca.key 8192; openssl req -new -x509 -extensions v3_ca -
key ca.key -out ca.crt -days 365
```

3. Set up the CA directory structure:

```
mkdir -m 0700 /root/CA /root/CA/certs /root/CA/crl /root/CA/newcerts /root/
CA/private
```

4. Move the CA key to `/root/CA/private` and the CA certificate to `/root/CA/certs`.

```
mv ca.key /root/CA/private;mv ca.crt /root/CA/certs
```

5. Add required files:

```
touch /root/CA/index.txt; echo 1000 >> /root/CA/serial
```

6. Set permissions on the `ca.key`:

```
chmod 0400 /root/ca/private/ca.key
```

7. Open the OpenSSL configuration file:

```
vi /etc/pki/tls/openssl.cnf
```

8. Change the directory paths to match your environment:

```
[ CA_default ]
```

```

dir            = /root/CA                # Where everything is kept
certs         = /root/CA/certs          # Where the issued certs are
  kept
crl_dir       = /root/CA/crl            # Where the issued crl are kept
database      = /root/CA/index.txt     # database index file.
#unique_subject = no                   # Set to 'no' to allow creation
  of                                     # several certificates with same
  subject.
new_certs_dir = /root/CA/newcerts      # default place for new certs.

certificate   = /root/CA/cacert.pem    # The CA certificate
serial        = /root/CA/serial        # The current serial number
crlnumber     = /root/CA/crlnumber     # the current crl number
# must be commented out to leave
  a V1 CRL
crl           = $dir/crl.pem           # The current CRL
private_key   = /root/CA/private/akey.pem # The private key
RANDFILE      = /root/CA/private/.rand # private random number file

x509_extensions = usr_cert            # The extensions to add to the cert

```

9. Save the changes and restart OpenSSL:

Example of setting up OpenSSL internal CA:

```

openssl genrsa -out ca.key 8192; openssl req -new -x509 -extensions v3_ca -key
ca.key -out ca.crt -days 365
Generating RSA private key, 8192 bit long modulus
.....++
.....++
e is 65537 (0x10001)
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:California
Locality Name (eg, city) [Default City]:Palo Alto
Organization Name (eg, company) [Default Company Ltd]:Hortonworks
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:nn
Email Address []:it@hortonworks.com

mkdir -m 0700 /root/CA /root/CA/certs /root/CA/crl /root/CA/newcerts /root/CA/
private
ls /root/CA
certs crl newcerts private

```

3.3.2.3. Create and Import Self-Signed Certificate

In small Hadoop Clusters, such as a cluster for demonstration, you can use self-signed certificates for SSL. Using self-signed certificates is less secure, for some services, such as Knox, to work with SSL the key must be imported into the Java CA store.

1. Create a self-signed certificate:

```
cd $SERVER_KEY_LOCATION ; keytool -genkey -alias $hostname -keyalg
RSA -keysize 1024 -dname CN=Hortonworks,OU=hw,O=hw,L=paloalto,ST=
ca,C=us -keypass $SERVER_KEYPASS_PASSWORD -keystore $KEYSTORE_FILE -
storepass $SERVER_STOREPASS_PASSWORD
```

2. Export the certificate to a file.

```
cd $SERVER_KEY_LOCATION ; keytool -export -alias $hostname
-keystore $KEYSTORE_FILE -rfc -file $CERTIFICATE_NAME -
storepass $TRUSTSTORE_PASSWORD
```

3. On each host, run the following command to export a certreq file from the host's keystore:

```
keytool -keystore keystore -alias `hostname -s` -certreq -file $host.cert -
storepass $hoststorekey -keypass $hostkey
```

4. For example, to create self-signed certificates for the Hadoop SSL Keystore Factory on the local host run:

```
mkdir -p /etc/security/serverKeys ; mkdir -p /etc/security/clientKeys
cd /etc/security/serverKeys ; keytool -genkey -alias $hostname -keyalg RSA
-keysize 1024 -dname "CN=Hortonworks,OU=hw,O=hw,L=paloalto,ST=ca,C=us" -
keypass changeit -keystore keystore.jks -storepass changeit
cd /etc/security/serverKeys ; keytool -export -alias $hostname -keystore
keystore.jks -rfc -file $hostname.jks -storepass changeit
```

Import the certificates:

```
keytool -keystore $hostkeystore -storepass $hoststorekey -alias `hostname -
s` -import -file $host.signed -keypass $hostkey
```

Push the keystore and truststore files back out to the other nodes in the cluster (along with the Hadoop configuration files and SSL Server and Client configurations), the first time you will need to restart affected cluster services. After, you can push only the keystore and truststore files and they are automatically reloaded.

3.3.2.4. Use a CA Signed Certificate

1. Run the following command to create a self-signing rootCA and import the rootCA into client truststore:

```
openssl genrsa -out $clusterCA.key 2048
openssl req -x509 -new -key $clusterCA.key -days 300 -out $clusterCA
keytool -importcert -alias $clusterCA -file $clusterCA -
keystore $clustertruststore -storepass $clustertruststorekey
```



Note

Ensure that the `ssl-client.xml` on every host configure to use this `'$clustertrust'` store. When configuring with Hive point to this file, and for all other services install the certificate in the Java truststore.

2. For each host, sign certreq file with the rootCA:

```
openssl x509 -req -CA $clusterCA.pem -CAkey $clusterCA.key -in $host.cert -
out $host.signed -days 300 -CAcreateserial
```

3. On each host, import both rootCA and the signed cert back in:

```
keytool -keystore $hostkeystore -storepass $hoststorekey -alias $clusterCA -
import -file cluser1CA.pem
keytool -keystore $hostkeystore -storepass $hoststorekey -alias `hostname` -
s` -import -file $host.signed -keypass $hostkey
```

3.3.2.5. Install Certificates in the Hadoop SSL Keystore Factory

The MapReduce, YARN, and HDFS use the Hadoop SSL Keystore Factory to manage SSL Certificates. This factory uses a common directory for server keystore and client truststore. The Hadoop SSL Keystore Factory allows you to use CA certificates or self-signed certificates managed in their own stores.

1. Create a directory for the server and client stores.

```
mkdir -p $SERVER_KEY_LOCATION ; mkdir -p $CLIENT_KEY_LOCATION
```

2. Import the server certificate from each node into the HTTP Factory truststore.

```
cd $SERVER_KEY_LOCATION ; keytool -import -noprompt -alias $remote-
hostname -file $remote-hostname.jks -keystore $TRUSTSTORE_FILE -
storepass $SERVER_TRUSTSTORE_PASSWORD
```

3. Create a single truststore file containing the public key from all certificates, by importing the public key for each CA or from each self-signed certificate pair:

```
keytool -import -noprompt -alias $host -file $CERTIFICATE_NAME -keystore
$ALL_JKS -storepass $CLIENT_TRUSTSTORE_PASSWORD
```

4. Copy the keystore and truststores to every node in the cluster.

5. Validate the common truststore file on all hosts.

```
keytool -list -v -keystore $ALL_JKS -storepass $CLIENT_TRUSTSTORE_PASSWORD
```

6. Set permissions and ownership on the keys:

```
chgrp -R $YARN_USER:hadoop $SERVER_KEY_LOCATION
chgrp -R $YARN_USER:hadoop $CLIENT_KEY_LOCATION
chmod 755 $SERVER_KEY_LOCATION
chmod 755 $CLIENT_KEY_LOCATION
chmod 440 $KEYSTORE_FILE
chmod 440 $TRUSTSTORE_FILE
chmod 440 $CERTIFICATE_NAME
chmod 444 $ALL_JKS
```



Note

The complete path of the \$SERVER_KEY_LOCATION and the CLIENT_KEY_LOCATION from the root directory /etc must be owned by the \$YARN_USER user and the hadoop group.

3.3.3. Enable SSL for WebHDFS, MapReduce Shuffle, and YARN

This section explains how to set up SSL for WebHDFS, YARN and MapReduce. Before you begin ensure that the SSL certificate is properly configured, including the keystore and truststore that will be used by WebHDFS, MapReduce, and YARN. HDP supports the following SSL modes:

- One-way SSL: SSL client validates the server identity only.
- Mutual authentication (2WAY SSL): Both the server and clients validate each others identity. 2WAY SSL can cause performance delays and is difficult to set up and maintain.



Note

In order to access SSL enabled HDP Services through the Knox Gateway, additional configuration on the Knox Gateway is required, see [Apache Knox Gateway Administrator Guide](#), [Gateway Security](#), [Configure Wire Encryption](#).

To enable one-way SSL set the following properties and restart all services:

1. Set the following property values (or add the properties if required) in `core-site.xml`:
 - `hadoop.ssl.require.client.cert=false`
 - `hadoop.ssl.hostname.verifier=DEFAULT`
 - `hadoop.ssl.keystores.factory.class=org.apache.hadoop.security.ssl.FileBasedKeyStoresFactor`
 - `hadoop.ssl.server.conf=ssl-server.xml`
 - `hadoop.ssl.client.conf=ssl-client.xml`



Note

Specify the `hadoop.ssl.server.conf` and `hadoop.ssl.client.conf` values as the relative or absolute path to Hadoop SSL Keystore Factory configuration files. If you specify only the file name, put the files in the same directory as the `core-site.xml`.

2. Set the following properties (or add the properties if required) in `hdfs-site.xml`:
 - `dfs.http.policy=$Policy`
 - `dfs.client.https.need-auth=true` (optional for mutual client/server certificate validation)
 - `dfs.datanode.https.address=$hostname:50475`
 - `dfs.namenode.https-address=$hostname:50470`

where `$Policy` is either:

- HTTP_ONLY: Service is provided only on HTTP
- HTTPS_ONLY: Service is provided only on HTTPS
- HTTP_AND_HTTPS: Service is provided both on HTTP and HTTPS

3. Set the following properties in `mapred-site.xml`:

```
mapreduce.jobhistory.http.policy=HTTPS_ONLY
mapreduce.jobhistory.webapp.https.address=<JHS>:<JHS_HTTPS_PORT>
```

4. Set the following properties in `yarn-site.xml`:

```
yarn.http.policy=HTTPS_ONLY
yarn.log.server.url=https://<JHS>:<JHS_HTTPS_PORT>/jobhistory/logs
yarn.resourcemanager.webapp.https.address=<RM>:<RM_HTTPS_PORT>
yarn.nodemanager.webapp.https.address=0.0.0.0:<NM_HTTPS_PORT>
```

5. Create a `ssl-server.xml` file for the Hadoop SSL Keystore Factory:

- a. Copy the example SSL Server configuration file and modify the settings for your environment:

```
cp /etc/hadoop/conf/ssl-server.xml.example /etc/hadoop/conf/ssl-server.xml
```

- b. Configure the server ssl properties:

Table 3.3. Configuration Properties in `ssl-server.xml`

Property	Default Value	Description
<code>ssl.server.keystore.type</code>	JKS	The type of the keystore, JKS = Java Keystore, the de-facto standard in Java
<code>ssl.server.keystore.location</code>	None	The location of the keystore file
<code>ssl.server.keystore.password</code>	None	The password to open the keystore file
<code>ssl.server.truststore.type</code>	JKS	The type of the trust store
<code>ssl.server.truststore.location</code>	None	The location of the truststore file
<code>ssl.server.truststore.password</code>	None	The password to open the truststore

For example:

```
<property>
  <name>ssl.server.truststore.location</name>
  <value>/etc/security/serverKeys/truststore.jks</value>
  <description>Truststore to be used by NN and DN. Must be specified.
</description>
</property>

<property>
  <name>ssl.server.truststore.password</name>
  <value>changeit</value>
  <description>Optional. Default value is "".
</description>
</property>
```

```

<property>
  <name>ssl.server.truststore.type</name>
  <value>jks</value>
  <description>Optional. The keystore file format, default value is
  "jks".
  </description>
</property>

<property>
  <name>ssl.server.truststore.reload.interval</name>
  <value>10000</value>
  <description>Truststore reload check interval, in milliseconds.
  Default value is 10000 (10 seconds).
  </description>
</property>

<property>
  <name>ssl.server.keystore.location</name>
  <value>/etc/security/serverKeys/keystore.jks</value>
  <description>Keystore to be used by NN and DN. Must be specified.
  </description>
</property>

<property>
  <name>ssl.server.keystore.password</name>
  <value>changeit</value>
  <description>Must be specified.
  </description>
</property>

<property>
  <name>ssl.server.keystore.keypassword</name>
  <value>changeit</value>
  <description>Must be specified.
  </description>
</property>

<property>
  <name>ssl.server.keystore.type</name>
  <value>jks</value>
  <description>Optional. The keystore file format, default value is
  "jks".
  </description>
</property>

```

6. Create a ssl-client.xml file for the Hadoop SSL Keystore Factory:

a. Copy the client truststore example file:

```
cp /etc/hadoop/conf/ssl-server.xml.example /etc/hadoop/conf/ssl-server.xml
```

b. Configure the client trust store values:

```
ssl.client.truststore.location=/etc/security/clientKeys/all.jks
ssl.client.truststore.password=clientTrustStorePassword
ssl.client.truststore.type=jks
```

7. Copy the configuration files (core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml, ssl-server.xml, and ssl-client.xml), including the ssl-server

and ssl-client store files if the Hadoop SSL Keystore Factory uses its own keystore and truststore files, to all nodes in the cluster.

8. Restart services on all nodes in the cluster.

3.3.4. Enable SSL on Oozie

The default SSL configuration makes all Oozie URLs use HTTPS except for the JobTracker callback URLs. This simplifies the configuration because no changes are required outside of Oozie. Oozie inherently does not trust the callbacks, they are used as hints.



Note

Before you begin ensure that the SSL certificate has been generated and properly configured. By default Oozie uses the user default keystore. In order to access SSL enabled HDP Services through the Knox Gateway, additional configuration on the Knox Gateway is required, see [Apache Knox Gateway Administrator Guide, Gateway Security, Configure Wire Encryption](#).

1. If Oozie server is running, stop Oozie.
2. Change the Oozie environment variables for HTTPS if required:
 - OOOZIE_HTTPS_PORT set to Oozie HTTPS port. The default value is 11443.
 - OOOZIE_HTTPS_KEYSTORE_FILE set to the keystore file that contains the certificate information. Default value `${HOME}/.keystore`, that is the home directory of the Oozie user.
 - OOOZIE_HTTPS_KEYSTORE_PASS set to the password of the keystore file. Default value password.



Note

The see [Oozie Environment Setup](#) for more details.

3. Run the following command to enable SSL on Oozie:

```
su -l oozie -c "oozie-setup.sh prepare-war -secure"
```

4. Start the Oozie server.



Note

To revert back to un-secured HTTP, run the following command:

```
su -l oozie -c "oozie-setup.sh prepare-war"
```

3.3.4.1. Configure Oozie HCatalogJob Properties

Integrate Oozie Hcatalog by adding following property to `oozie-hcatalog job.properties`. For example if you are using Ambari, set the properties as:

```
hadoop.rpc.protection=privacy
```




Note

This property is in addition to any properties you must set for secure clusters.

3.3.5. Enable SSL on WebHBase and the HBase REST API

Perform the following task to enable SSL on WebHBase and HBase REST API.



Note

In order to access SSL enabled HDP Services through the Knox Gateway, additional configuration on the Knox Gateway is required, see [Apache Knox Gateway Administrator Guide, Gateway Security, Configure Wire Encryption](#).

1. Verify that the HBase REST API is running, on the HBase Master run:

```
curl http://localhost:60080/
```

If the rest daemon is not running on that port, run the following command to start it:

```
sudo /usr/lib/hbase/bin/hbase-daemon.sh start rest -p 60080
```

2. Create and install an SSL certificate for HBase, for example to use a self-signed certificate:

- a. Create an HBase keystore:

```
su -l hbase -c "keytool -genkey -alias hbase -keyalg RSA -keysize 1024 -keystore hbase.jks"
```

- b. Export the certificate:

```
su -l hbase -c "keytool -exportcert -alias hbase -file certificate.cert -keystore hbase.jks"
```

- c. Add certificate to the Java keystore:

- If you are not root run:

```
sudo keytool -import -alias hbase -file certificate.cert -keystore /usr/jdk64/jdk1.7.0_45/jre/lib/security/cacerts
```

- If you are root:

```
keytool -import -alias hbase -file certificate.cert -keystore /usr/jdk64/jdk1.7.0_45/jre/lib/security/cacerts
```

3. Add the following properties to the `hbase-site.xml` configuration file on each node in your HBase cluster:

```
<property>
  <name>hbase.rest.ssl.enabled</name>
  <value>true</value>
</property>

<property>
  <name>hbase.rest.ssl.keystore.store</name>
```

```

    <value>/path/to/keystore</value>
  </property>

  <property>
    <name>hbase.rest.ssl.keystore.password</name>
    <value>$keystore-password</value>
  </property>

  <property>
    <name>hbase.rest.ssl.keystore.keypassword</name>
    <value>$key-password</value>
  </property>

```

- Restart all HBase nodes in the cluster.



Note

When using a self-signed certificate, manually add the certificate to the JVM truststore on all HBase clients.

3.3.6. Enable SSL on HiveServer2

When using HiveServer2 without Kerberos authentication, you can enable SSL.



Note

In order to access SSL enabled HDP Services through the Knox Gateway, additional configuration on the Knox Gateway is required, see [Apache Knox Gateway Administrator Guide, Gateway Security, Configure Wire Encryption](#).

Perform the following steps on the HiveServer2:

- Run the following command to create a keystore for hiveserver2::

```
keytool -genkey -alias hbase -keyalg RSA -keysize 1024 -keystore hbase.jks
```

- Edit the `hive-site.xml`, set the following properties to enable SSL:

```

<property>
  <name>hive.server2.enable.SSL</name>
  <value>>true</value>
  <description></description>
</property>

<property>
  <name>hive.server2.keystore.path</name>
  <value>$keystore-file-path</value>
  <description></description>
</property>

<property>
  <name>hive.server2.keystore.password</name>
  <value>$keystore-file-password</value>
  <description></description>
</property>

```

- On the client-side, specify SSL settings for Beeline or JDBC client as follows:

```
jdbc:hive2://$host:$port/$database;ssl=true;sslTrustStore=$path-to-truststore;trustStorePassword=$password
```

3.4. Connect to SSL Enabled Components

This section explains how to connect to SSL enabled HDP Components:

- [Section 3.4.1, "Connect to SSL Enabled HiveServer2 using JDBC " \[55\]](#)
- [Section 3.4.2, "Connect to SSL Enabled Oozie Server" \[55\]](#)



Note

In order to access SSL enabled HDP Services through the Knox Gateway, additional configuration on the Knox Gateway is required, see [Apache Knox Gateway Administrator Guide, Gateway Security, Configure Wire Encryption](#).

3.4.1. Connect to SSL Enabled HiveServer2 using JDBC

HiveServer2 implemented encryption with the Java SASL protocol's quality of protection (QOP) setting that allows data moving between a HiveServer2 over JDBC and a JDBC client to be encrypted.

From the JDBC client specify `sasl.sop` as part of the JDBC-Hive connection string, for example `jdbc:hive://hostname/dbname;sasl.qop=auth-int`. For more information on connecting to Hive, see [Data Integration Services with HDP, Using Apache Hive: Hive ODBC and JDBC Drivers](#).



Tip

See [HIVE-4911](#) for more details on this enhancement.

3.4.2. Connect to SSL Enabled Oozie Server

On every Oozie client system, follow the instructions for the type of certificate used in your environment.

3.4.2.1. Use a Self-signed Certificate from Oozie Java Clients

When using a self-signed certificate, you must first install the certificate before the Oozie client can connect to the server.

1. Install the certificate in the keychain:
 - a. Copy or download the `.cert` file onto the client machine.
 - b. Run the following command (as root) to import the certificate into the JRE's keystore:

```
sudo keytool -import -alias tomcat -file path/to/certificate.cert -keystore $JRE_cacerts
```

Where `$JRE_cacerts` is the path to the JRE's certs file. It's location may differ depending on the Operating System, but its typically called cacerts and located at `$JAVA_HOME/lib/security/cacerts`. It can be under a different directory in `$JAVA_HOME`. The default password is `changeit`.

Java programs, including the Oozie client, can now connect to the Oozie Server using the self-signed certificate.

2. In the connection strings change HTTP to HTTPS, for example, replace `http://oozie.server.hostname:11000/oozie` with `https://oozie.server.hostname:11443/oozie`.

Java does not automatically redirect HTTP addresses to HTTPS.

3.4.2.2. Connect to Oozie from Java Clients

In the connection strings change HTTP to HTTPS and adjust the port, for example, replace `http://oozie.server.hostname:11000/oozie` with `https://oozie.server.hostname:11443/oozie`.

Java does not automatically redirect HTTP addresses to HTTPS.

3.4.2.3. Connect to Oozie from a Web Browser

Use `https://oozie.server.hostname:11443/oozie` though most browsers should automatically redirect you if you use `http://oozie.server.hostname:11000/oozie`.

When using a Self-Signed Certificate, your browser warns you that it can't verify the certificate. Add the certificate as an exception.