cloudera®

Cloudera Security

Cloudera, Inc.
1001 Page Mill Road, Bldg 3
Palo Alto, CA 94304
info@cloudera.com
US: 1-888-789-1488
Intl: 1-650-362-0488
www.cloudera.com

Release Information

Version: Cloudera Enterprise 5.13.x
Date: October 12, 2017

Table of Contents

About this Guide

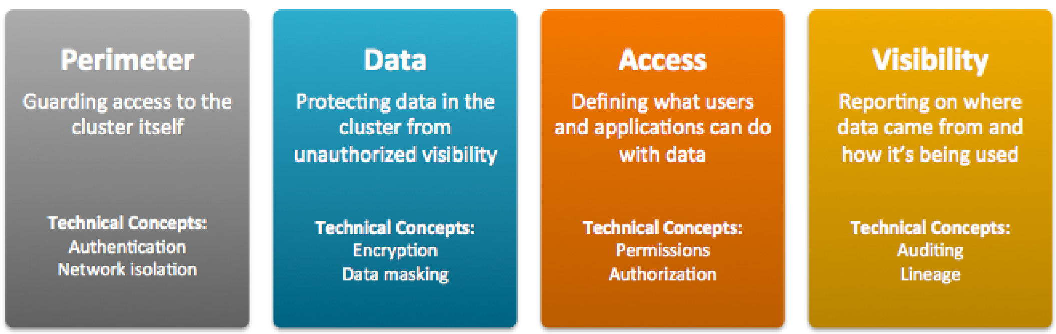
This guide is intended for system administrators who want to secure a cluster using data encryption, user authentication, and authorization techniques. It provides conceptual overviews and how-to information about setting up various Hadoop components for optimal security, including how to setup a gateway to restrict access. This guide assumes that you have basic knowledge of Linux and systems administration practices, in general.

Cloudera Security Overview

As a system designed to support vast amounts and types of data, Cloudera clusters must meet ever-evolving security requirements imposed by regulating agencies, governments, industries, and the general public. Cloudera clusters comprise both Hadoop core and ecosystem components, all of which must be protected from a variety of threats to ensure the confidentiality, integrity, and availability of all the cluster's services and data. This Security Guide aims to help you configure your Cloudera cluster for secure operations. It starts by introducing Security Requirements on page 14 and Security Levels on page 14 and then provides overviews of core security mechanisms, as follows:

Security Requirements

Goals for data management systems, such as confidentiality, integrity, and availability, require that the system be secured across several dimensions. These can be characterized in terms of both general operational goals and technical concepts, as shown in the figure below:



- **Perimeter** Access to the cluster must be protected from a variety of threats coming from internal and external networks and from a variety of actors. Network isolation can be provided by proper configuration of firewalls, routers, subnets, and the proper use of public and private IP addresses, for example. **Authentication** mechanisms ensure that people, processes, and applications properly identify themselves to the cluster and prove they are who they say they are, before gaining access to the cluster.
- **Data** Data in the cluster must always be protected from unauthorized exposure. Similarly, communications between the nodes in the cluster must be protected. **Encryption** mechanisms ensure that even if network packets are intercepted or hard-disk drives are physically removed from the system by bad actors, the contents are not usable.
- **Access** Access to any specific service or item of data within the cluster must be specifically granted. **Authorization** mechanisms ensure that once users have authenticated themselves to the cluster, they can only see the data and use the processes to which they have been granted specific permission.
- **Visibility** Visibility means that the history of data changes is transparent and capable of meeting data governance policies. **Auditing** mechanisms ensure that all actions on data and its lineage—source, changes over time, and so on—are documented as they occur.

Securing the cluster to meet specific organizational goals involves using security features inherent to the Hadoop ecosystem as well as using external security infrastructure. The various security mechanisms can be applied in a range of levels.

Security Levels

The figure below shows the range of security levels that can be implemented for a Cloudera cluster, from non-secure (0) to most secure (3). As the sensitivity and volume of data on the cluster increases, so should the security level you choose for the cluster.

With level 3 security, your Cloudera cluster is ready for full compliance with various industry and regulatory mandates and is ready for audit when necessary. The table below describes the levels in more detail:

Level	Security	Characteristics
0	Non-secure	No security configured. Non-secure clusters should never be used in production environments because they are vulnerable to any and all attacks and exploits.
1	Minimal	Configured for authentication, authorization, and auditing. Authentication is first configured to ensure that users and services can access the cluster only after proving their identities. Next, authorization mechanisms are applied to assign privileges to users and user groups. Auditing procedures keep track of who accesses the cluster (and how).
2	More	Sensitive data is encrypted. Key management systems handle encryption keys. Auditing has been setup for data in metastores. System metadata is reviewed and updated regularly. Ideally, cluster has been setup so that lineage for any data object can be traced (data governance).
3	Most	The secure enterprise data hub (EDH) is one in which all data, both data-at-rest and data-in-transit, is encrypted and the key management system is fault-tolerant. Auditing mechanisms comply with industry, government, and regulatory standards (PCI, HIPAA, NIST, for example), and extend from the EDH to the other systems that integrate with it. Cluster administrators are well-trained, security procedures have been certified by an expert, and the cluster can pass technical review.

Authentication Concepts

Authentication is a basic security requirement for any computing environment. Users, services, and other processes must prove their identity—authenticate—to the system before they can use any of the resources they are authorized to use. Authentication and authorization work hand-in-hand to ensure that system resources, from processing cycles to the data itself, are protected from unauthorized use.

In a properly configured cluster, several different mechanisms work together to provide authentication and authorization for users and other services:

- Kerberos provides strong authentication—meaning it relies on a cryptographic intermediary process to handle communications between a requesting client and server, and goes well beyond simple password-based authentication. Kerberos is recommended for production clusters. Cloudera clusters can be configured to use MIT Kerberos or Microsoft Active Directory. Microsoft implemented Kerberos as its authentication protocol (replacing NT LAN Manager) over a decade ago.
- On each host operating system underlying each node in a cluster, local Linux `user:group` accounts are created during installation of Cloudera Server and CDH services. In order to apply what is essentially a per-node authentication and authorization mechanism consistently across all the nodes of a cluster, local `user:group`

accounts are mapped to user accounts and groups in an LDAP-compliant directory service, such as Active Directory or OpenLDAP. See Configuring LDAP Group Mappings on page 343 for details.

  - To facilitate the authentication process from each host system (node in the cluster) to the LDAP directory, Cloudera recommends using additional software mechanisms such as SSSD (Systems Security Services Daemon) or Centrify Server Suite. See the Centrify guide Identity and Access management for Cloudera for details.

- Authorization is handled in many different ways, from access control lists (ACLs), to HDFS extended ACLs, to role-based access controls (RBAC) using Sentry. See Authorization Concepts on page 29 for more information.

Cloudera Manager Server, CDH services, Cloudera Navigator, Apache Hive, Hue, and Impala (which all support external clients) can use Kerberos for authentication. In addition, Kerberos credentials can be stored and managed in the LDAP-compliant identity service. This section provides some of the conceptual background. See Configuring Authentication for details.

Basic Kerberos Concepts

This section describes how Hadoop uses Kerberos principals and keytabs for user authentication. It also briefly describes how Hadoop uses delegation tokens to authenticate jobs at execution time, to avoid overwhelming the KDC with authentication requests for each job.

Kerberos Principals

A user in Kerberos is called a ***principal***, which is made up of three distinct components: the primary, instance, and realm. A ***Kerberos principal*** is used in a Kerberos-secured system to represent a unique identity. The first component of the principal is called the ***primary***, or sometimes the user component. The primary component is an arbitrary string and may be the operating system username of the user or the name of a service. The primary component is followed by an optional section called the ***instance***, which is used to create principals that are used by users in special roles or to define the host on which a service runs, for example. An instance, if it exists, is separated from the primary by a slash and then the content is used to disambiguate multiple principals for a single user or service. The final component of the principal is the realm. The ***realm*** is similar to a domain in DNS in that it logically defines a related group of objects, although rather than hostnames as in DNS, the Kerberos realm defines a group of principals . Each realm can have its own settings including the location of the KDC on the network and supported encryption algorithms. Large organizations commonly create distinct realms to delegate administration of a realm to a group within the enterprise. Realms, by convention, are written in uppercase characters.

Kerberos assigns tickets to Kerberos principals to enable them to access Kerberos-secured Hadoop services. For the Hadoop daemon principals, the principal names should be of the format `username/fully.qualified.domain.name@YOUR-REALM.COM`. In this guide, *username* in the `username/fully.qualified.domain.name@YOUR-REALM.COM` principal refers to the username of an existing Unix account that is used by Hadoop daemons, such as `hdfs` or `mapred`. Human users who want to access the Hadoop cluster also need to have Kerberos principals; in this case, *`username`* refers to the username of the user's Unix account, such as `joe` or `jane`. Single-component principal names (such as `joe@YOUR-REALM.COM`) are acceptable for client user accounts. Hadoop does not support more than two-component principal names.

Kerberos Keytabs

A ***keytab*** is a file containing pairs of Kerberos principals and an encrypted copy of that principal's key. A keytab file for a Hadoop daemon is unique to each host since the principal names include the hostname. This file is used to authenticate a principal on a host to Kerberos without human interaction or storing a password in a plain text file. Because having access to the keytab file for a principal allows one to act as that principal, access to the keytab files should be tightly secured. They should be readable by a minimal set of users, should be stored on local disk, and should not be included in host backups, unless access to those backups is as secure as access to the local host.

Delegation Tokens

Users in a Hadoop cluster authenticate themselves to the NameNode using their Kerberos credentials. However, once the user is authenticated, each job subsequently submitted must also be checked to ensure it comes from an authenticated user. Since there could be a time gap between a job being submitted and the job being executed, during which the user could have logged off, user credentials are passed to the NameNode using delegation tokens that can be used for authentication in the future.

Delegation tokens are a secret key shared with the NameNode, that can be used to impersonate a user to get a job executed. While these tokens can be renewed, new tokens can only be obtained by clients authenticating to the NameNode using Kerberos credentials. By default, delegation tokens are only valid for a day. However, since jobs can last longer than a day, each token specifies a JobTracker as a *renewer* which is allowed to renew the delegation token once a day, until the job completes, or for a maximum period of 7 days. When the job is complete, the JobTracker requests the NameNode to cancel the delegation token.

Token Format

The NameNode uses a random `masterKey` to generate delegation tokens. All active tokens are stored in memory with their expiry date (`maxDate`). Delegation tokens can either expire when the current time exceeds the expiry date, or, they can be canceled by the owner of the token. Expired or canceled tokens are then deleted from memory. The `sequenceNumber` serves as a unique ID for the tokens. The following section describes how the Delegation Token is used for authentication.

```
TokenID = {ownerID, renewerID, issueDate, maxDate, sequenceNumber}
TokenAuthenticator = HMAC-SHA1(masterKey, TokenID)
Delegation Token = {TokenID, TokenAuthenticator}
```

Authentication Process

To begin the authentication process, the client first sends the TokenID to the NameNode. The NameNode uses this TokenID and the `masterKey` to once again generate the corresponding TokenAuthenticator, and consequently, the Delegation Token. If the NameNode finds that the token already exists in memory, and that the current time is less than the expiry date (`maxDate`) of the token, then the token is considered valid. If valid, the client and the NameNode will then authenticate each other by using the TokenAuthenticator that they possess as the secret key, and MD5 as the protocol. Since the client and NameNode do not actually exchange TokenAuthenticators during the process, even if authentication fails, the tokens are not compromised.

Token Renewal

Delegation tokens must be renewed periodically by the designated renewer (`renewerID`). For example, if a JobTracker is the designated renewer, the JobTracker will first authenticate itself to the NameNode. It will then send the token to be authenticated to the NameNode. The NameNode verifies the following information before renewing the token:

- The JobTracker requesting renewal is the same as the one identified in the token by `renewerID`.
- The TokenAuthenticator generated by the NameNode using the `TokenID` and the `masterKey` matches the one previously stored by the NameNode.
- The current time must be less than the time specified by `maxDate`.

If the token renewal request is successful, the NameNode sets the new expiry date to `min(current time+renew period, maxDate)`. If the NameNode was restarted at any time, it will have lost all previous tokens from memory. In this case, the token will be saved to memory once again, this time with a new expiry date. Hence, designated renewers must renew all tokens with the NameNode after a restart, and before relaunching any failed tasks.

A designated renewer can also revive an expired or canceled token as long as the current time does not exceed `maxDate`. The NameNode cannot tell the difference between a token that was canceled, or has expired, and one that was erased from memory due to a restart, since only the `masterKey` persists in memory. The `masterKey` must be updated regularly.

Types of Kerberos Deployments

Kerberos provides strong security benefits including capabilities that render intercepted authentication packets unusable by an attacker. It virtually eliminates the threat of impersonation by never sending a user's credentials in cleartext over the network. Several components of the Hadoop ecosystem are converging to use Kerberos authentication with the option to manage and store credentials in LDAP or AD. Microsoft's Active Directory (AD) is an LDAP directory that also provides Kerberos authentication for added security. Before you configure Kerberos on your cluster, ensure you have a working KDC (MIT KDC or Active Directory), set up. You can then use Cloudera Manager's Kerberos wizard to automate several aspects of configuring Kerberos authentication on your cluster.

Without Kerberos enabled, Hadoop only checks to ensure that a user and their group membership is valid in the context of HDFS. However, it makes no effort to verify that the user is who they say they are.

With Kerberos enabled, users must first authenticate themselves to a Kerberos Key Distribution Centre (KDC) to obtain a valid Ticket-Granting-Ticket (TGT). The TGT is then used by Hadoop services to verify the user's identity. With Kerberos, a user is not only authenticated on the system they are logged into, but they are also authenticated to the network. Any subsequent interactions with other services that have been configured to allow Kerberos authentication for user access, are also secured.

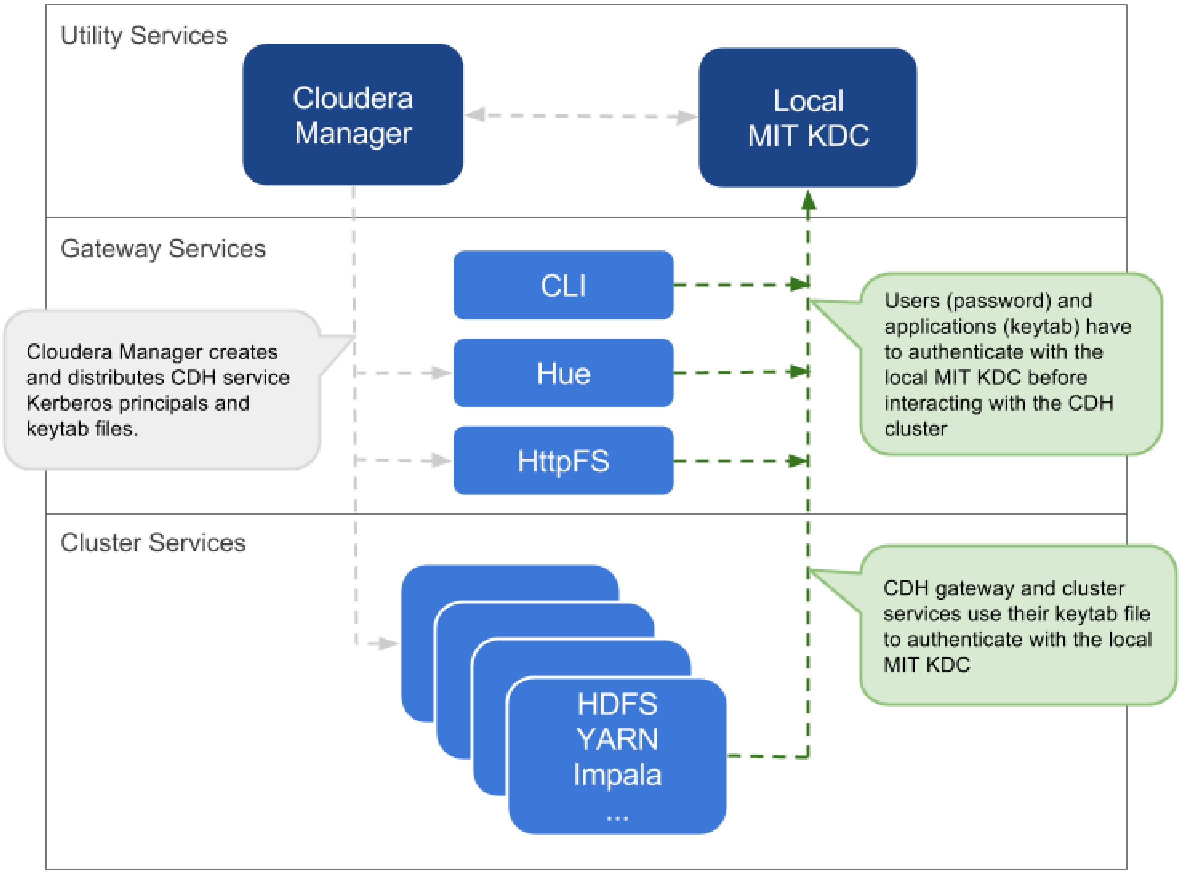
This section describes the architectural options that are available for deploying Hadoop security in enterprise environments. Each option includes a high-level description of the approach along with a list of pros and cons. There are three options currently available:

Local MIT KDC

This approach uses an MIT KDC that is local to the cluster. Users and services will have to authenticate with this local KDC before they can interact with the CDH components on the cluster.

Architecture Summary:

- An MIT KDC and a distinct Kerberos realm is deployed locally to the CDH cluster. The local MIT KDC is typically deployed on a Utility host. Additional replicated MIT KDCs for high-availability are optional.
- All cluster hosts must be configured to use the local MIT Kerberos realm using the `krb5.conf` file.
- All **service and user principals** must be created in the local MIT KDC and Kerberos realm.
- The local MIT KDC will authenticate both the service principals (using keytab files) and user principals (using passwords).
- Cloudera Manager connects to the local MIT KDC to create and manage the principals for the CDH services running on the cluster. To do this Cloudera Manager uses an admin principal and keytab that is created during the security setup. This step has been automated by the Kerberos wizard. Instructions for manually creating the Cloudera Manager admin principal are provided in the Cloudera Manager security documentation.
- Typically, the local MIT KDC administrator is responsible for creating all other user principals. If you use the Kerberos wizard, Cloudera Manager will create these principals and associated keytab files for you.



Pros	Cons
The authentication mechanism is isolated from the rest of the enterprise.	This mechanism is not integrated with central authentication system.
This is fairly easy to setup, especially if you use the Cloudera Manager Kerberos wizard that automates creation and distribution of service principals and keytab files.	User and service principals must be created in the local MIT KDC, which can be time-consuming.
	The local MIT KDC can be a single point of failure for the cluster unless replicated KDCs can be configured for high-availability.
	The local MIT KDC is yet another authentication system to manage.

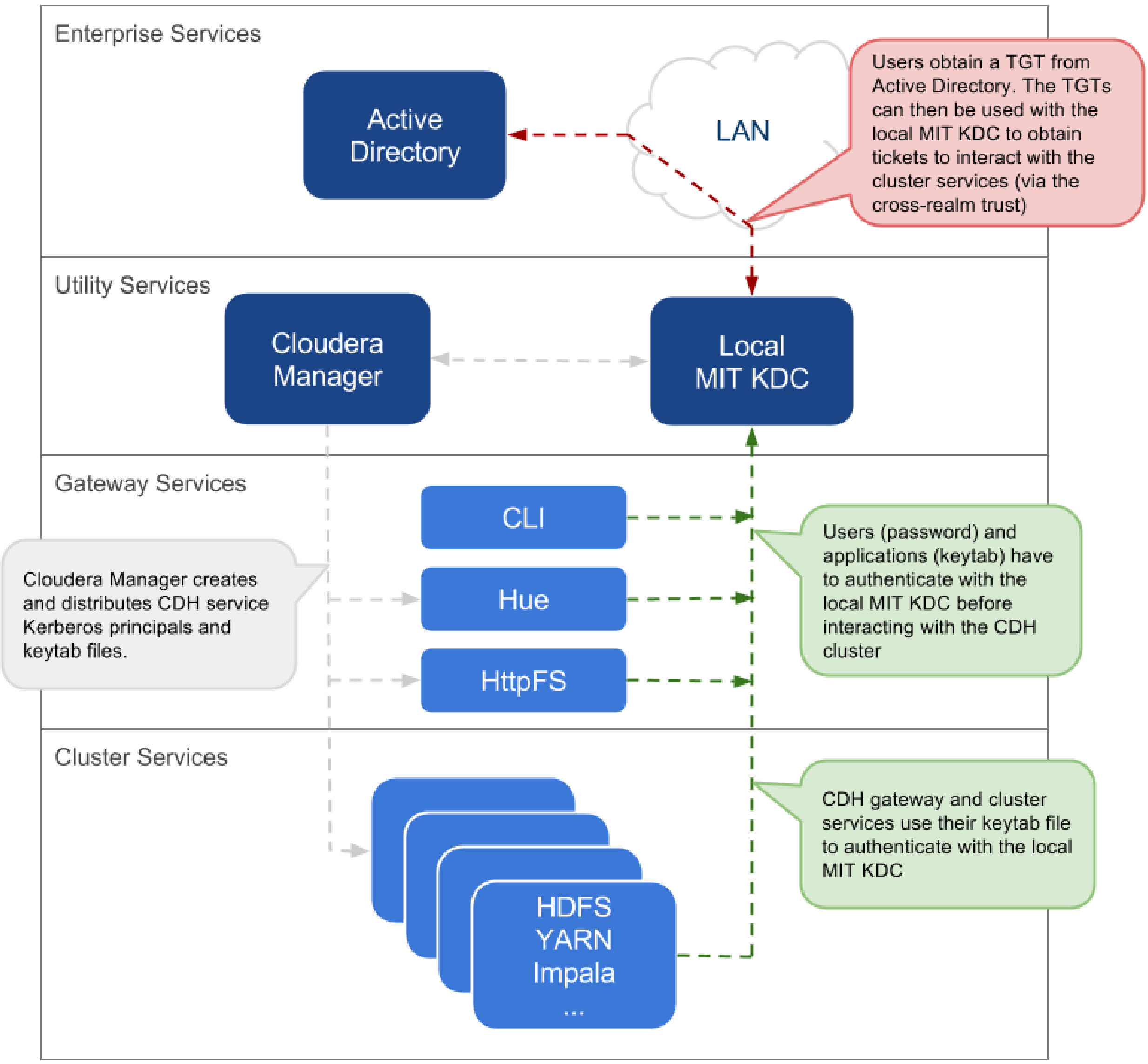
Local MIT KDC with Active Directory Integration

This approach uses an MIT KDC and Kerberos realm that is local to the cluster. However, Active Directory stores the user principals that will access the cluster in a central realm. Users will have to authenticate with this central AD realm to obtain TGTs before they can interact with CDH services on the cluster. Note that CDH service principals reside only in the local KDC realm.

Architecture Summary:

- An MIT KDC and a distinct Kerberos realm is deployed locally to the CDH cluster. The local MIT KDC is typically deployed on a Utility host and additional replicated MIT KDCs for high-availability are optional.
- All cluster hosts are configured with both Kerberos realms (local and central AD) using the `krb5.conf` file. The default realm should be the local MIT Kerberos realm.

- **Service principals** should be created in the local MIT KDC and the local Kerberos realm. Cloudera Manager connects to the local MIT KDC to create and manage the principals for the CDH services running on the cluster. To do this, Cloudera Manager uses an admin principal and keytab that is created during the security setup. This step has been automated by the Kerberos wizard. Instructions for manually creating the Cloudera Manager admin principal are provided in the Cloudera Manager security documentation.
- A one-way, cross-realm trust must be set up from the local Kerberos realm to the central AD realm containing the **user principals** that require access to the CDH cluster. There is no need to create the service principals in the central AD realm and no need to create user principals in the local realm.



Pros	Cons
The local MIT KDC serves as a shield for the central Active Directory from the many hosts and services in a CDH cluster. Service restarts in a large cluster create many simultaneous authentication requests. If Active Directory is unable to handle the spike in load, then the cluster can effectively cause a distributed denial of service (DDOS) attack.	The local MIT KDC can be a single point of failure (SPOF) for the cluster. Replicated KDCs can be configured for high-availability.

Pros	Cons
This is fairly easy to setup, especially if you use the Cloudera Manager Kerberos wizard that automates creation and distribution of service principals and keytab files. Active Directory administrators will only need to be involved to configure the cross-realm trust during setup.	The local MIT KDC is yet another authentication system to manage.
Integration with central Active Directory for user principal authentication results in a more complete authentication solution.	
Allows for incremental configuration. Hadoop security can be configured and verified using local MIT KDC independently of integrating with Active Directory.	

Direct to Active Directory

This approach uses the central Active Directory as the KDC. No local KDC is required. Before you decide upon an AD KDC deployment, make sure you are aware of the following possible ramifications of that decision.

Considerations when using an Active Directory KDC

Performance:

As your cluster grows, so will the volume of Authentication Service (AS) and Ticket Granting Service (TGS) interaction between the services on each cluster server. Consider evaluating the volume of this interaction against the Active Directory domain controllers you have configured for the cluster before rolling this feature out to a production environment. If cluster performance suffers, over time it might become necessary to dedicate a set of AD domain controllers to larger deployments. Cloudera recommends you use a dedicated AD instance for every 100 nodes in your cluster. However, note that this recommendation may not apply to high-volume clusters, or cases where the AD host is also being used for LDAP lookups.

Network Proximity:

By default, Kerberos uses UDP for client/server communication. Often, AD services are in a different network than project application services such as Hadoop. If the domain controllers supporting a cluster for Kerberos are not in the same subnet, or they're separated by a firewall, consider using the `udp_preference_limit = 1` setting in the `[libdefaults]` section of the `krb5.conf` used by cluster services. Cloudera strongly recommends *against* using AD domain controller (KDC) servers that are separated from the cluster by a WAN connection, as latency in this service will significantly impact cluster performance.

Process:

Troubleshooting the cluster's operations, especially for Kerberos-enabled services, will need to include AD administration resources. Evaluate your organizational processes for engaging the AD administration team, and how to escalate in case a cluster outage occurs due to issues with Kerberos authentication against AD services. In some situations it might be necessary to enable Kerberos event logging to address desktop and KDC issues within windows environments.

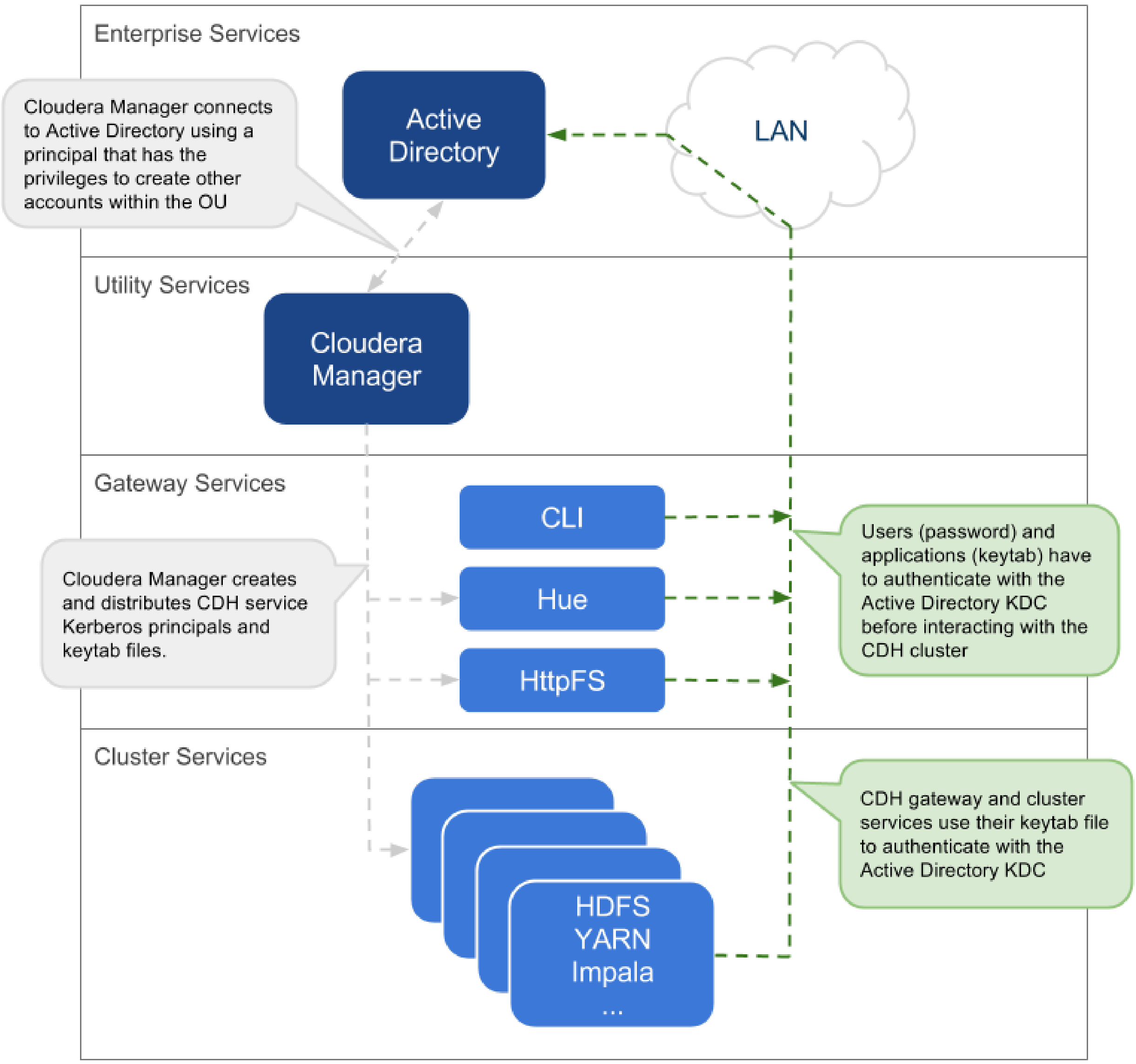
Also note that if you decommission any Cloudera Manager roles or nodes, the related AD accounts will need to be deleted manually. This is required because Cloudera Manager will not delete existing entries in Active Directory.

Architecture Summary

- All **service and user principals** are created in the Active Directory KDC.
- All cluster hosts are configured with the central AD Kerberos realm using `krb5.conf`.
- Cloudera Manager connects to the Active Directory KDC to create and manage the principals for the CDH services running on the cluster. To do this, Cloudera Manager uses a principal that has the privileges to create other accounts within the given Organisational Unit (OU). This step has been automated by the Kerberos wizard.

Instructions for manually creating the Cloudera Manager admin principal are provided in the Cloudera Manager security documentation.

- All service and user principals are authenticated by the Active Directory KDC.



Note: If it is not possible to create the Cloudera Manager admin principal with the required privileges in the Active Directory KDC, then the CDH services principals will need to be created manually. The corresponding keytab files should then be stored securely on the Cloudera Manager Server host. Cloudera Manager's Custom Kerberos Keytab Retrieval script can be used to retrieve the keytab files from the local filesystem.

Identity Integration with Active Directory

A core requirement for enabling Kerberos security in the platform is that users have accounts on all cluster processing nodes. Commercial products such as Centrify or Quest Authentication Services (QAS) provide integration of all cluster hosts for user and group resolution to Active Directory. These tools support automated Kerberos authentication on login by users to a Linux host with AD. For sites not using Active Directory, or sites wanting to use an open source solution, the Site Security Services Daemon (SSSD) can be used with either AD or OpenLDAP compatible directory services and MIT Kerberos for the same needs.

For third-party providers, you may have to purchase licenses from the respective vendors. This procedure requires some planning as it takes time to procure these licenses and deploy these products on a cluster. Care should be taken to ensure that the identity management product does not associate the service principal names (SPNs) with the host principals when the computers are joined to the AD domain. For example, Centrify by default associates the HTTP SPN with the host principal. So the HTTP SPN should be specifically excluded when the hosts are joined to the domain.

You will also need to complete the following setup tasks in AD:

- **Active Directory Organizational Unit (OU) and OU user** - A separate OU in Active Directory should be created along with an account that has privileges to create additional accounts in that OU.
- **Enable SSL for AD** - Cloudera Manager should be able to connect to AD on the LDAPS (TCP 636) port.
- **Principals and Keytabs** - In a direct-to-AD deployment that is set up using the Kerberos wizard, by default, all required principals and keytabs will be created, deployed and managed by Cloudera Manager. However, if for some reason you cannot allow Cloudera Manager to manage your direct-to-AD deployment, then unique accounts should be manually created in AD for each service running on each host and keytab files must be provided for the same. These accounts should have the AD User Principal Name (UPN) set to `service/fqdn@REALM`, and the Service Principal Name (SPN) set to `service/fqdn`. The principal name in the keytab files should be the UPN of the account. The keytab files should follow the naming convention: `servicename_fqdn.keytab`. The following principals and keytab files must be created for each host they run on: Hadoop Users in Cloudera Manager and CDH on page 163.
- **AD Bind Account** - Create an AD account that will be used for LDAP bindings in Hue, Cloudera Manager and Cloudera Navigator.
- **AD Groups for Privileged Users** - Create AD groups and add members for the authorized users, HDFS admins and HDFS superuser groups.
  - Authorized users – A group consisting of all users that need access to the cluster
  - HDFS admins – Groups of users that will run HDFS administrative commands
  - HDFS super users – Group of users that require superuser privilege, that is, read/wwrite access to all data and directories in HDFS

    Putting regular users into the HDFS superuser group is *not* recommended. Instead, an account that administrators escalate issues to, should be part of the HDFS superuser group.
- **AD Groups for Role-Based Access to Cloudera Manager and Cloudera Navigator** - Create AD groups and add members to these groups so you can later configure role-based access to Cloudera Manager and Cloudera Navigator.
- **AD Test Users and Groups** - At least one existing AD user and the group that the user belongs to should be provided to test whether authorization rules work as expected.

Securing Keytab Distribution using TLS/SSL

The Kerberos keytab file is transmitted among the hosts in the Cloudera Manager cluster, between Cloudera Manager Server and Cloudera Manager Agent hosts. To keep this sensitive data secure, configure Cloudera Manager Server and the Cloudera Manager Agent hosts for encrypted communications using TLS/SSL. See Data in Transit Encryption (TLS/SSL) on page 179 for details.

Configuring Kerberos Authentication on a Cluster

Before you use the following sections to configure Kerberos on your cluster, ensure you have a working KDC (MIT KDC or Active Directory), set up.

- To use the Cloudera Manager wizard, see Enabling Kerberos Authentication Using the Wizard on page 50.
- For command-line configuration, see Enabling Kerberos Authentication Without the Wizard on page 67.

Authentication Mechanisms used by Cluster Components

Component or Product	Authentication Mechanism Supported
HDFS	Kerberos, SPNEGO (HttpFS)
MapReduce	Kerberos *(also see HDFS)*
YARN	Kerberos *(also see HDFS)*
Accumulo	Kerberos (partial)
Flume	Kerberos (starting CDH 5.4)
HBase	Kerberos *(HBase Thrift and REST clients must perform their own user authentication)*
HiveServer	None
HiveServer2	Kerberos, LDAP, Custom/pluggable authentication
Hive Metastore	Kerberos
Hue	Kerberos, LDAP, SAML, Custom/pluggable authentication
Impala	Kerberos, LDAP, SPNEGO (Impala Web Console)
Oozie	Kerberos, SPNEGO
Pig	Kerberos
Search	Kerberos, SPNEGO
Sentry	Kerberos
Spark	Kerberos
Sqoop	Kerberos
Sqoop2	Kerberos (starting CDH 5.4)
Zookeeper	Kerberos
Cloudera Manager	Kerberos, LDAP, SAML
Cloudera Navigator	Active Directory, OpenLDAP, SAML
Backup and Disaster Recovery	Same as Cloudera Manager.

Encryption Concepts

Encryption is a process that uses digital keys to encode various components—text, files, databases, passwords, applications, or network packets, for example—so that only the appropriate entity (user, system process, and so on) can decode (decrypt) the item and view, modify, or add to the data. Cloudera provides encryption mechanisms to protect data persisted to disk or other storage media (***data at rest encryption*** or simply, data encryption) and as it moves over the network (***data in transit encryption***).

Data encryption is mandatory in government, health, finance, education, and many other environments. For example, the Federal Information Security Management Act (FISMA) governs patient privacy concerns and the Payment Card Industry Data Security Standard (PCI DSS) regulates information security for credit-card processors. These are just two examples.

The vast quantity of data contained in Cloudera clusters, deployed using many different components, must nonetheless support whatever degree of privacy, confidentiality, and data integrity is required by the use case. The encryption mechanisms supported by Cloudera and discussed in this overview aim to do just that.

Protecting Data At-Rest

Protecting data at rest typically means encrypting the data when it is stored on disk and letting authorized users and processes—and only authorized users and processes—to decrypt the data when needed for the application or task at hand. With data-at-rest encryption, encryption keys must be distributed and managed, keys should be rotated or changed on a regular basis (to reduce the risk of having keys compromised), and many other factors complicate the process.

However, encrypting data alone may not sufficient. For example, administrators and others with sufficient privileges may have access to personally identifiable information (PII) in log files, audit data, or SQL queries. Depending on the specific use case—in hospital or financial environment, the PII may need to be redacted from all such files, to ensure that users with privileges on the logs and queries that might contain sensitive data are nonetheless unable to view that data when they should not.

Cloudera provides complementary approaches to encrypting data at rest, and provides mechanisms to mask PII in log files, audit data, and SQL queries.

Encryption Options Available

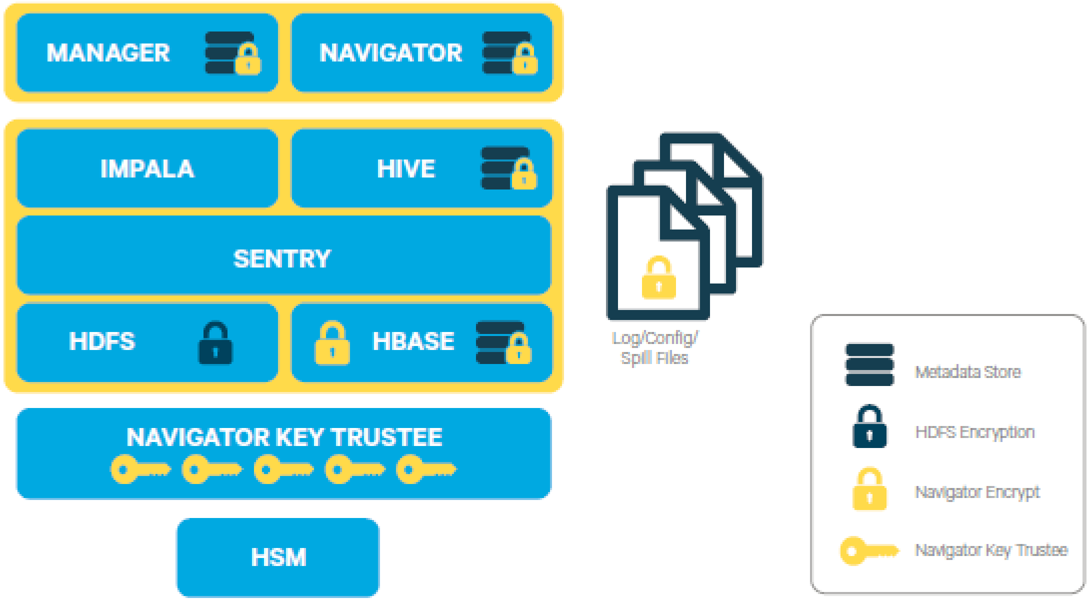
Cloudera provides several mechanisms to ensure that sensitive data is secure. CDH provides transparent HDFS encryption, ensuring that all sensitive data is encrypted before being stored on disk. HDFS encryption when combined with the enterprise-grade encryption key management of Navigator Key Trustee enables regulatory compliance for most enterprises. For Cloudera Enterprise, HDFS encryption can be augmented by Navigator Encrypt to secure metadata, in addition to data. Cloudera clusters that use these solutions run as usual and have very low performance impact, given that data nodes are encrypted in parallel. As the cluster grows, encryption grows with it.

Additionally, this transparent encryption is optimized for the Intel chipset for high performance. Intel chipsets include AES-NI co-processors, which provide special capabilities that make encryption workloads run extremely fast. Cloudera leverages the latest Intel advances for even faster performance. Additionally, HDFS Encryption and Navigator Encrypt feature separation of duties, preventing even IT Administrators and root users from accessing data that they are not authorized to see.

The Key Trustee KMS, used in conjunction with Key Trustee Server and Key HSM, provides HSM-based protection of stored key material. The Key Trustee KMS generates encryption zone key material locally on the KMS and then encrypts this key material using an HSM-generated key. Navigator HSM KMS services, in contrast, rely on the HSM for all encryption zone key generation and storage. When using the Navigator HSM KMS, encryption zone key material originates on the HSM and never leaves the HSM. This allows for the highest level of key isolation, but requires some overhead for network calls to the HSM for key generation, encryption and decryption operations. The Key Trustee KMS remains the recommended key management solution for HDFS encryption for most production scenarios.

The figure below shows an example deployment that uses:

- Cloudera Transparent HDFS Encryption to encrypt data stored on HDFS
- Navigator Encrypt for all other data (including metadata, logs, and spill data) associated with Cloudera Manager, Cloudera Navigator, Hive, and HBase
- Navigator Key Trustee for robust, fault-tolerant key management



In addition to applying encryption to the data layer of a Cloudera cluster, encryption can also be applied at the network layer, to encrypt communications among nodes of the cluster. See Encryption Mechanisms Overview on page 28 for more information.

Encryption does not prevent administrators with full access to the cluster from viewing sensitive data. To obfuscate sensitive data, including PII, the cluster can be configured for data redaction.

Data Redaction for Cloudera Clusters

Redaction is a process that obscures data. It can help organizations comply with industry regulations and standards, such as PCI (Payment Card Industry) and HIPAA, by obfuscating personally identifiable information (PII) so that is not usable except by those whose jobs require such access. For example, HIPAA legislation requires that patient PII not be available to anyone other than appropriate physician (and the patient), and that any patient's PII cannot be used to determine or associate an individual's identity with health data. Data redaction is one process that can help ensure this privacy, by transforming PII to meaningless patterns—for example, transforming U.S. social security numbers to XXX-XX-XXXX strings.

Data redaction works separately from Cloudera encryption techniques, which do not preclude administrators with full access to the cluster from viewing sensitive user data. It ensures that cluster administrators, data analysts, and others cannot see PII or other sensitive data that is not within their job domain and at the same time, it does not prevent users with appropriate permissions from accessing data to which they have privileges.

See How to Enable Sensitive Data Redaction for details.

Protecting Data In-Transit

For data-in-transit, implementing data protection and encryption is relatively easy. Wire encryption is built into the Hadoop stack, such as SSL, and typically does not require external systems. This data-in-transit encryption is built using session-level, one-time keys, by means of a session handshake with immediate and subsequent transmission. Thus, data-in-transit avoids much of the key management issues associated with data-at-rest due the temporal nature of the keys, but it does rely on proper authentication; a certificate compromise is an issue with authentication, but can compromise wire encryption. As the name implies, data-in-transit covers the secure transfer and intermediate storage of data. This applies to all process-to-process communication, within the same node or between nodes. There are three primary communication channels:

- **HDFS Transparent Encryption:** Data encrypted using HDFS Transparent Encryption is protected end-to-end. Any data written to and from HDFS can only be encrypted or decrypted by the client. HDFS does not have access to the unencrypted data or the encryption keys. This supports both, at-rest encryption as well as in-transit encryption.

- **Data Transfer:** The first channel is data transfer, including the reading and writing of data blocks to HDFS. Hadoop uses a SASL-enabled wrapper around its native direct TCP/IP-based transport, called `DataTransportProtocol`, to secure the I/O streams within an DIGEST-MD5 envelope (For steps, see How to Configure Encrypted Transport for HDFS Data on page 455). This procedure also employs secured HadoopRPC (see Remote Procedure Calls) for the key exchange. The HttpFS REST interface, however, does not provide secure communication between the client and HDFS, only secured authentication using SPNEGO.

  For the transfer of data between DataNodes during the shuffle phase of a MapReduce job (that is, moving intermediate results between the Map and Reduce portions of the job), Hadoop secures the communication channel with HTTP Secure (HTTPS) using Transport Layer Security (TLS). See Encrypted Shuffle and Encrypted Web UIs on page 220.

- **Remote Procedure Calls:** The second channel is system calls to remote procedures (RPC) to the various systems and frameworks within a Hadoop cluster. Like data transfer activities, Hadoop has its own native protocol for RPC, called HadoopRPC, which is used for Hadoop API client communication, intra-Hadoop services communication, as well as monitoring, heartbeats, and other non-data, non-user activity. HadoopRPC is SASL-enabled for secured transport and defaults to Kerberos and DIGEST-MD5 depending on the type of communication and security settings. For steps, see How to Configure Encrypted Transport for HDFS Data on page 455.
- **User Interfaces:** The third channel includes the various web-based user interfaces within a Hadoop cluster. For secured transport, the solution is straightforward; these interfaces employ HTTPS.

TLS/SSL Certificates Overview

Certificates can be signed in one three different ways:

Type	Usage Note
Public CA-signed certificates	**Recommended**. Using certificates signed by a trusted public CA simplifies deployment because the default Java client already trusts most public CAs. Obtain certificates from one of the trusted well-known (public) CAs, such as Symantec and Comodo, as detailed in Generate TLS Certificates on page 457
Internal CA-signed certificates	Obtain certificates from your organization's internal CA if your organization has its own. Using an internal CA can reduce costs (although cluster configuration may require establishing the trust chain for certificates signed by an internal CA, depending on your IT infrastructure). See How to Configure TLS Encryption for Cloudera Manager on page 456 for information about establishing trust as part of configuring a Cloudera Manager cluster.
Self-signed certificates	Not recommended for production deployments. Using self-signed certificates requires configuring each client to trust the specific certificate (in addition to generating and distributing the certificates). However, self-signed certificates are fine for non-production (testing or proof-of-concept) deployments. See How to Use Self-Signed Certificates for TLS on page 490 for details.

For more information on setting up SSL/TLS certificates, see Data in Transit Encryption (TLS/SSL) on page 179.

TLS/SSL Encryption for CDH Components

Cloudera recommends securing a cluster using Kerberos authentication before enabling encryption such as SSL on a cluster. If you enable SSL for a cluster that does not already have Kerberos authentication configured, a warning will be displayed.

Hadoop services differ in their use of SSL as follows:

- HDFS, MapReduce, and YARN daemons act as both SSL servers and clients.
- HBase daemons act as SSL servers only.
- Oozie daemons act as SSL servers only.
- Hue acts as an SSL client to all of the above.

Daemons that act as SSL servers load the keystores when starting up. When a client connects to an SSL server daemon, the server transmits the certificate loaded at startup time to the client, which then uses its truststore to validate the server's certificate.

For information on setting up SSL/TLS for CDH services, see Configuring TLS/SSL Encryption for CDH Services on page 198.

Data Protection within Hadoop Projects

The table below lists the various encryption capabilities that can be leveraged by CDH components and Cloudera Manager.

Project	**Encryption for Data-in-Transit**	**Encryption for Data-at-Rest** **(HDFS Encryption + Navigator Encrypt + Navigator Key Trustee)**
HDFS	SASL (RPC), SASL (DataTransferProtocol)	Yes
MapReduce	SASL (RPC), HTTPS (encrypted shuffle)	Yes
YARN	SASL (RPC)	Yes
Accumulo	Partial - Only for RPCs and Web UI (Not directly configurable in Cloudera Manager)	Yes
Flume	TLS (Avro RPC)	Yes
HBase	SASL - For web interfaces, inter-component replication, the HBase shell and the REST, Thrift 1 and Thrift 2 interfaces	Yes
HiveServer2	SASL (Thrift), SASL (JDBC), TLS (JDBC, ODBC)	Yes
Hue	TLS	Yes
Impala	TLS or SASL between impalad and clients, but not between daemons	
Oozie	TLS	Yes
Pig	N/A	Yes
Search	TLS	Yes
Sentry	SASL (RPC)	Yes
Spark	None	Yes
Sqoop	Partial - Depends on the RDBMS database driver in use	Yes
Sqoop2	Partial - You can encrypt the JDBC connection depending on the RDBMS database driver	Yes
ZooKeeper	SASL (RPC)	No
Cloudera Manager	TLS - Does not include monitoring	Yes
Cloudera Navigator	TLS - *Also see Cloudera Manager*	Yes
Backup and Disaster Recovery	TLS - *Also see Cloudera Manager*	Yes

Encryption Mechanisms Overview

Data at rest and data in transit encryption function at different technology layers of the cluster:

Layer	Description
Application	Applied by the HDFS client software, HDFS Transparent Encryption lets you encrypt specific folders contained in HDFS. To securely store the required encryption keys, Cloudera recommends using Cloudera Navigator Key Trustee Server in conjunction with HDFS encryption. See Enabling HDFS Encryption Using Navigator Key Trustee Server on page 241 for details. Data stored temporarily on the local filesystem outside HDFS by CDH components (including Impala, MapReduce, YARN, or HBase) can also be encrypted. See Configuring Encryption for Data Spills on page 334 for details.
Operating System	At the Linux OS filesystem layer, encryption can be applied to an entire volume. For example, Cloudera Navigator Encrypt can encrypt data inside and outside HDFS, such as temp/spill files, configuration files, and databases that store metadata associated with a CDH cluster. Navigator Encrypt requires a license for Cloudera Navigator and must be configured to use Navigator Key Trustee Server.
Network	Network communications between client processes and server processes (HTTP, RPC, or TCP/IP services) can be encrypted using industry-standard TLS/SSL as detailed in Data in Transit Encryption (TLS/SSL) on page 179.

Here are some good starting places for more information about encryption for Cloudera clusters:

- Data at rest encryption
  - Cloudera Navigator Encrypt
  - HDFS Transparent Encryption
  - How to Configure Encryption for Amazon S3
- Data in transit encryption
  - How to Configure TLS Encryption for Cloudera Manager (a complete step-by-step guide)

Authorization Concepts

Authorization is one of the fundamental security requirements of any computing environment. Its goal is to ensure that only the appropriate people or processes can access, view, use, control, or change specific resources, services, or data. In any cluster deployed to meet specific workloads using various CDH components (Hive, HDFS, Impala, and so on), different authorization mechanisms can ensure that only authorized users or processes can access data, systems, and other resources as needed. Ideally, authorization mechanisms can leverage the authentication mechanisms, so that when users login to a system—a cluster, for example—they are transparently authorized based on their identity across the system for the applications, data, and other resources they are authorized to use.

For example, Cloudera CDH clusters can be configured to leverage the user and group accounts that exist in the organization's Active Directory (or other LDAP-accessible directory) instance.

The various possible configurations and integrations are discussed later in this guide.

Authorization Mechanisms in Hadoop

Hadoop supports several authorization mechanisms, including:

- Traditional POSIX-style permissions on files and directories. Each directory and file has a single owner and group with basic permissions that can be set to read, write, execute (at the file level). Directories have an additional permission that enables access to child directories.
- Access Control Lists (ACL) for management of services and resources. For example, Apache HBase uses ACLs to authorize various operations (`READ, WRITE, CREATE, ADMIN`) by column, column family, and column family qualifier. HBase ACLs are granted and revoked to users and groups. Fine-grained permissions can be applied to

HDFS files using HDFS Extended ACLs on page 342 which enable setting different permissions for specific named users and named groups.

- Role-Based Access Control (RBAC) for certain services with advanced access controls to data. Apache Sentry provides role-based access control for the cluster, including access to Hive, Impala, and Solr services. Cloudera recommends using the database-backed Sentry Service to configure permissions rather than using policy files. However, both approaches are supported. SeeAuthorization With Apache Sentry on page 346 for more information.

POSIX Permissions

Most services running on Hadoop clusters, such as the command-line interface (CLI) or client applications that use Hadoop API, directly access data stored within HDFS. HDFS uses POSIX-style permissions for directories and files; each directory and file is assigned a single owner and group. Each assignment has a basic set of permissions available; file permissions are read, write, and execute, and directories have an additional permission to determine access to child directories.

Ownership and group membership for a given HDFS asset determines a user's privileges. If a given user fails either of these criteria, they are denied access. For services that may attempt to access more than one file, such as MapReduce, Cloudera Search, and others, data access is determined separately for each file access attempt. File permissions in HDFS are managed by the NameNode.

Access Control Lists

Hadoop also maintains general access controls for the services themselves in addition to the data within each service and in HDFS. Service access control lists (ACL) are typically defined within the global hadoop-policy.xml file and range from NameNode access to client-to-DataNode communication. In the context of MapReduce and YARN, user and group identifiers form the basis for determining permission for job submission or modification.

In addition, with MapReduce and YARN, jobs can be submitted using queues controlled by a scheduler, which is one of the components comprising the resource management capabilities within the cluster. Administrators define permissions to individual queues using ACLs. ACLs can also be defined on a job-by-job basis. Like HDFS permissions, local user accounts and groups must exist on each executing server, otherwise the queues will be unusable except by superuser accounts.

Apache HBase also uses ACLs for data-level authorization. HBase ACLs authorize various operations (READ, WRITE, CREATE, ADMIN) by column, column family, and column family qualifier. HBase ACLs are granted and revoked to both users and groups. Local user accounts are required for proper authorization, similar to HDFS permissions.

Apache ZooKeeper also maintains ACLs to the information stored within the DataNodes of a ZooKeeper data tree.

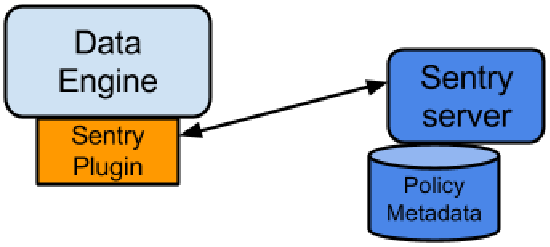
Role-Based Access Control with Apache Sentry

For fine-grained access to data accessible using schema—that is, data structures described by the Apache Hive Metastore and used by computing engines like Hive and Impala, as well as collections and indices within Cloudera Search—CDH supports Apache Sentry, which offers a role-based privilege model for this data and its given schema. Apache Sentry is a role-based authorization module for Hadoop. Sentry lets you configure explicit privileges for authenticated users and applications on a Hadoop cluster. Sentry is integrated natively with Apache Hive, Apache Solr, HDFS (for Hive table data), Hive Metastore/HCatalog, and Impala.

Sentry comprises a pluggable authorization engine for Hadoop components. It lets you define authorization rules to validate requests for access to resources from users or applications. The authorization holds the rules and privileges, but it's the engines that apply the rules at runtime.

Architecture Overview

Sentry Components



The subsystems nvolved in the authorization process:

- **Sentry Server**—An RPC server that manages authorization metadata stored in a database. It supports interfaces to securely retrieve and manipulate the metadata.
- **Data Engine**

  This is a data processing application such as Hive or Impala that needs to authorize access to data or metadata resources. The data engine loads the Sentry plugin and all client requests for accessing resources are intercepted and routed to the Sentry plugin for validation.

- **Sentry Plugin**

  The Sentry plugin runs in the data engine. It offers interfaces to manipulate authorization metadata stored in the Sentry Server, and includes the authorization policy engine that evaluates access requests using the authorization metadata retrieved from the server.

Key Concepts

- Authentication - Verifying credentials to reliably identify a user
- Authorization - Limiting the user's access to a given resource
- User - Individual identified by underlying authentication system
- Group - A set of users, maintained by the authentication system
- Privilege - An instruction or rule that allows access to an object
- Role - A set of privileges; a template to combine multiple access rules
- Authorization models - Defines the objects to be subject to authorization rules and the granularity of actions allowed. For example, in the SQL model, the objects can be databases or tables, and the actions are `SELECT`, `INSERT`, and `CREATE`. For the Search model, the objects are indexes, collections and documents; the access modes are query and update.

User Identity and Group Mapping

Sentry relies on underlying authentication systems such as Kerberos or LDAP to identify the user. It also uses the group mapping mechanism configured in Hadoop to ensure that Sentry sees the same group mapping as other components of the Hadoop ecosystem.

Consider users Alice and Bob who belong to an Active Directory (AD) group called `finance-department`. Bob also belongs to a group called `finance-managers`. In Sentry, you first create roles and then grant privileges to these roles. For example, you can create a role called Analyst and grant SELECT on tables Customer and Sales to this role.

The next step is to join these authentication entities (users and groups) to authorization entities (roles). This can be done by granting the Analyst role to the `finance-department` group. Now Bob and Alice who are members of the `finance-department` group get SELECT privilege to the Customer and Sales tables.

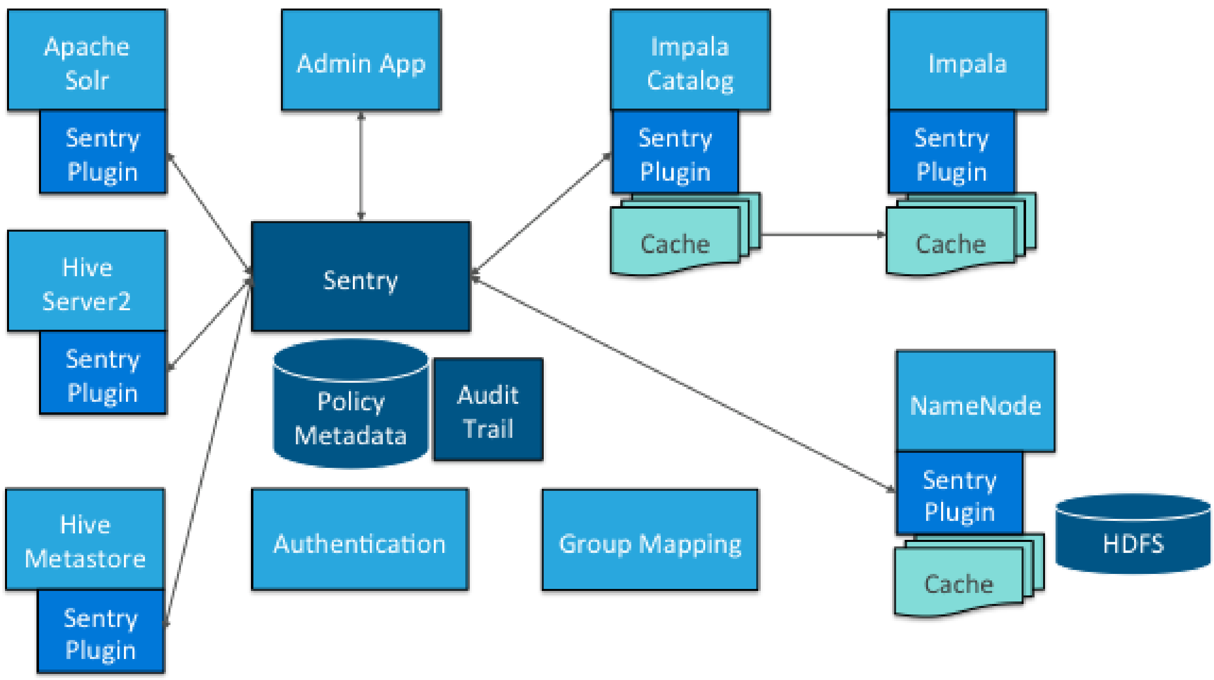
Role-based access control (RBAC) is a powerful mechanism to manage authorization for a large set of users and data objects in a typical enterprise. New data objects get added or removed, users join, move, or leave organisations all the time. RBAC makes managing this a lot easier. Hence, as an extension of the discussed previously, if Carol joins the

Finance Department, all you need to do is add her to the `finance-department` group in AD. This will give Carol access to data from the Sales and Customer tables.

Unified Authorization

Another important aspect of Sentry is the unified authorization. The access control rules once defined, work across multiple data access tools. For example, being granted the Analyst role in the previous example will allow Bob, Alice, and others in the `finance-department` group to access table data from SQL engines such as Hive and Impala, as well as using MapReduce, Pig applications or metadata access using HCatalog.

Sentry Integration with the Hadoop Ecosystem



As illustrated above, Apache Sentry works with multiple Hadoop components. At the heart you have the Sentry Server which stores authorization metadata and provides APIs for tools to retrieve and modify this metadata securely.

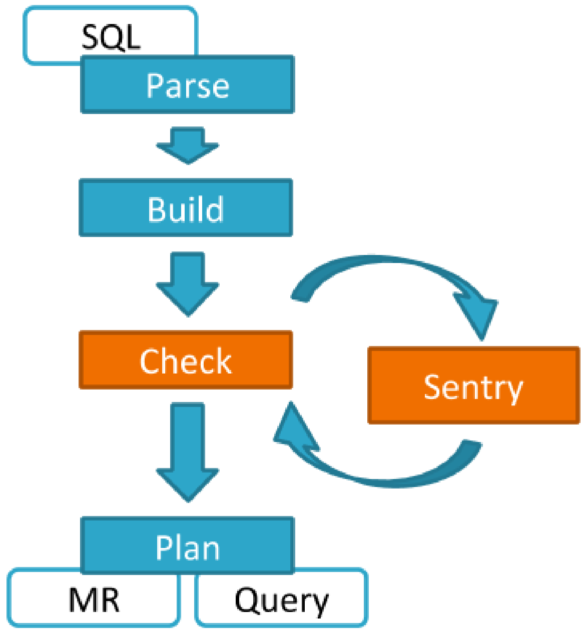
Note that the Sentry Server only facilitates the metadata. The actual authorization decision is made by a policy engine which runs in data processing applications such as Hive or Impala. Each component loads the Sentry plugin which includes the service client for dealing with the Sentry service and the policy engine to validate the authorization request.

Hive and Sentry

Consider an example where Hive gets a request to access an object in a certain mode by a client. If Bob submits the following Hive query:

```
select * from production.sales
```

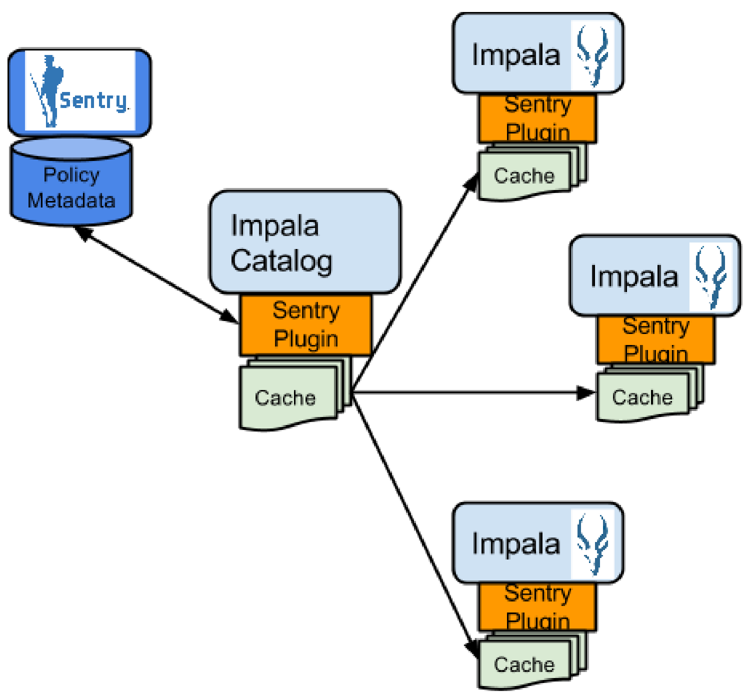
Hive will identify that user Bob is requesting SELECT access to the Sales table. At this point Hive will ask the Sentry plugin to validate Bob's access request. The plugin will retrieve Bob's privileges related to the Sales table and the policy engine will determine if the request is valid.



Hive works with both, the Sentry service and policy files. Cloudera recommends you use the Sentry service which makes it easier to manage user privileges. For more details and instructions, see The Sentry Service on page 351 *or* Sentry Policy File Authorization on page 392.

Impala and Sentry

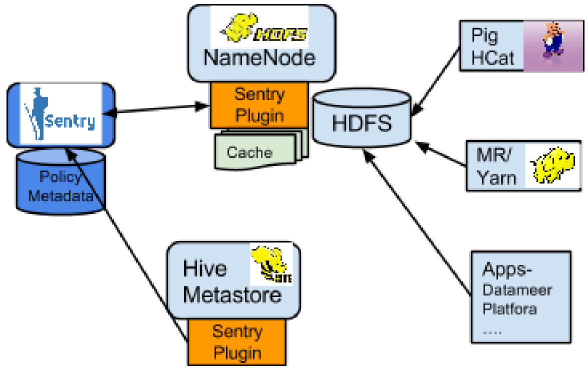
Authorization processing in Impala is similar to that in Hive. The main difference is caching of privileges. Impala's Catalog server manages caching schema metadata and propagating it to all Impala server nodes. This Catalog server caches Sentry metadata as well. As a result, authorization validation in Impala happens locally and much faster.



For detailed documentation, see Enabling Sentry Authorization for Impala on page 415.

Sentry-HDFS Synchronization

Sentry-HDFS authorization is focused on Hive warehouse data - that is, any data that is part of a table in Hive or Impala. The real objective of this integration is to expand the same authorization checks to Hive warehouse data being accessed from any other components such as Pig, MapReduce or Spark. At this point, this feature does not replace HDFS ACLs. Tables that are not associated with Sentry will retain their old ACLs.



The mapping of Sentry privileges to HDFS ACL permissions is as follows:

- SELECT privilege -> Read access on the file.
- INSERT privilege -> Write access on the file.
- ALL privilege -> Read and Write access on the file.

The NameNode loads a Sentry plugin that caches Sentry privileges as well Hive metadata. This helps HDFS to keep file permissions and Hive tables privileges in sync. The Sentry plugin periodically polls the Sentry and Metastore to keep the metadata changes in sync.

For example, if Bob runs a Pig job that is reading from the Sales table data files, Pig will try to get the file handle from HDFS. At that point the Sentry plugin on the NameNode will figure out that the file is part of Hive data and overlay Sentry privileges on top of the file ACLs. As a result, HDFS will enforce the same privileges for this Pig client that Hive would apply for a SQL query.

For HDFS-Sentry synchronization to work, you *must* use the Sentry service, not policy file authorization. See Synchronizing HDFS ACLs and Sentry Permissions on page 387, for more details.

Search and Sentry

Sentry can apply a range of restrictions to various Search tasks, such accessing data or creating collections. These restrictions are consistently applied, regardless of the way users attempt to complete actions. For example, restricting access to data in a collection restricts that access whether queries come from the command line, from a browser, or through the admin console.

With Search, Sentry stores its privilege policies in a policy file (for example, sentry-provider.ini) which is stored in an HDFS location such as `hdfs://ha-nn-uri/user/solr/sentry/sentry-provider.ini`.

Sentry with Search does not support multiple policy files for multiple databases. However, you must use a separate policy file for each Sentry-enabled service. For example, Hive and Search were using policy file authorization, using a combined Hive and Search policy file would result in an invalid configuration and failed authorization on both services.

> **Note:** While Hive and Impala are compatible with the database-backed Sentry service, Search still uses Sentry's policy file authorization. Note that it is possible for a single cluster to use both, the Sentry service (for Hive and Impala as described above) and Sentry policy files (for Solr).

For detailed documentation, see Configuring Sentry Authorization for Cloudera Search on page 425.

Authorization Administration

The Sentry Server supports APIs to securely manipulate roles and privileges. Both Hive and Impala support SQL statements to manage privileges natively. Sentry assumes that HiveServer2 and Impala run as superusers, usually called `hive` and `impala`. To initiate top-level permissions for Sentry, an admin must login as a superuser.You can use either Beeline or the Impala shell to execute the following sample statement:

```
GRANT ROLE Analyst TO GROUP finance-managers
```

Using Hue to Manage Sentry Permissions

Hue supports a Security app to manage Sentry authorization. This allows users to explore and change table permissions. Here is a video blog that demonstrates its functionality.

Integration with Authentication Mechanisms for Identity Management

Like many distributed systems, Hadoop projects and workloads often consist of a collection of processes working in concert. In some instances, the initial user process conducts authorization throughout the entirety of the workload or job's lifecycle. But for processes that spawn additional processes, authorization can pose challenges. In this case, the spawned processes are set to execute as if they were the authenticated user, that is, setuid, and thus only have the privileges of that user. The overarching system requires a mapping to the authenticated principal and the user account must exist on the local host system for the setuid to succeed.

Important:

- Cloudera strongly recommends *against* using Hadoop's `LdapGroupsMapping` provider. `LdapGroupsMapping` should only be used in cases where OS-level integration is not possible. Production clusters require an identity provider that works well with all applications, not just Hadoop. Hence, often the preferred mechanism is to use tools such as SSSD, VAS or Centrify to replicate LDAP groups.
- Cloudera does not support the use of Winbind in production environments. Winbind uses an inefficient approach to user/group mapping, which may lead to low performance or cluster failures as the size of the cluster, and the number of users and groups increases.

Irrespective of the mechanism used, user/group mappings must be applied consistently across all cluster hosts for ease with maintenance.

System and Service Authorization - Several Hadoop services are limited to inter-service interactions and are not intended for end-user access. These services do support authentication to protect against unauthorized or malicious users. However, any user or, more typically, another service that has login credentials and can authenticate to the service is authorized to perform all actions allowed by the target service. Examples include ZooKeeper, which is used by internal systems such as YARN, Cloudera Search, and HBase, and Flume, which is configured directly by Hadoop administrators and thus offers no user controls.

The authenticated Kerberos principals for these “system” services are checked each time they access other services such as HDFS, HBase, and MapReduce, and therefore must be authorized to use those resources. Thus, the fact that Flume does not have an explicit authorization model does not imply that Flume has unrestricted access to HDFS and other services; the Flume service principals still must be authorized for specific locations of the HDFS file system. Hadoop administrators can establish separate system users for a services such as Flume to segment and impose access rights to only the parts of the file system for a specific Flume application.

Authorization within Hadoop Projects

Project	Authorization Capabilities
HDFS	File Permissions, Sentry*
MapReduce	File Permissions, Sentry*
YARN	File Permissions, Sentry*
Accumulo	

Project	Authorization Capabilities
Flume	None
HBase	HBase ACLs
HiveServer2	File Permissions, Sentry
Hue	Hue authorization mechanisms (assigning permissions to Hue apps)
Impala	Sentry
Oozie	ACLs
Pig	File Permissions, Sentry*
Search	File Permissions, Sentry
Sentry	N/A
Spark	File Permissions, Sentry*
Sqoop	N/A
Sqoop2	None
ZooKeeper	ACLs
Cloudera Manager	Cloudera Manager roles
Cloudera Navigator	Cloudera Navigator roles
Backup and Disaster Recovery	N/A

* Sentry HDFS plug-in; when enabled, Sentry enforces its own access permissions over files that are part of tables defined in the Hive Metastore.

Auditing and Data Lineage Concepts

Organizations of all kinds want to understand where data in their clusters is coming from and how it is used. Cloudera Navigator Data Management component is a fully integrated data management and security tool for the Hadoop platform that has been designed to meet compliance, data governance, and auditing needs of global enterprises. Without Cloudera Navigator, Hadoop clusters rely primarily on log files for auditing. However, log files are not an enterprise-class real-time auditing or monitoring solution. For example, log files can be corrupted by a system crash during a write commit.

Cloudera Navigator Data Management

Cloudera Navigator captures a complete and immutable record of all system activity. An audit trail can be used to determine the particulars—the who, what, where, and when—of a data breach or attempted breach. Auditing can be used to not only identify a rogue administrator who deleted user data, for example, but can also be used to recover data from a backup. Enterprises that must prove they are in compliance with HIPAA (Health Insurance Portability and Accountability Act), PCI (Payment Card Industry Data Security Standard), or other regulations associated with sensitive or personally identifiable data (PII) are required to produce auditing records when asked by government or other officials, such as banking regulators.

Auditing also serves to provide a historical record of data and context for data forensics. Data stewards and curators can use auditing reports to determine consumption and use patterns across various data sets by different user communities, for optimizing data access.

This section provides a brief overview of functionality of Cloudera Navigator. For complete details, see Cloudera Navigator Data Management.

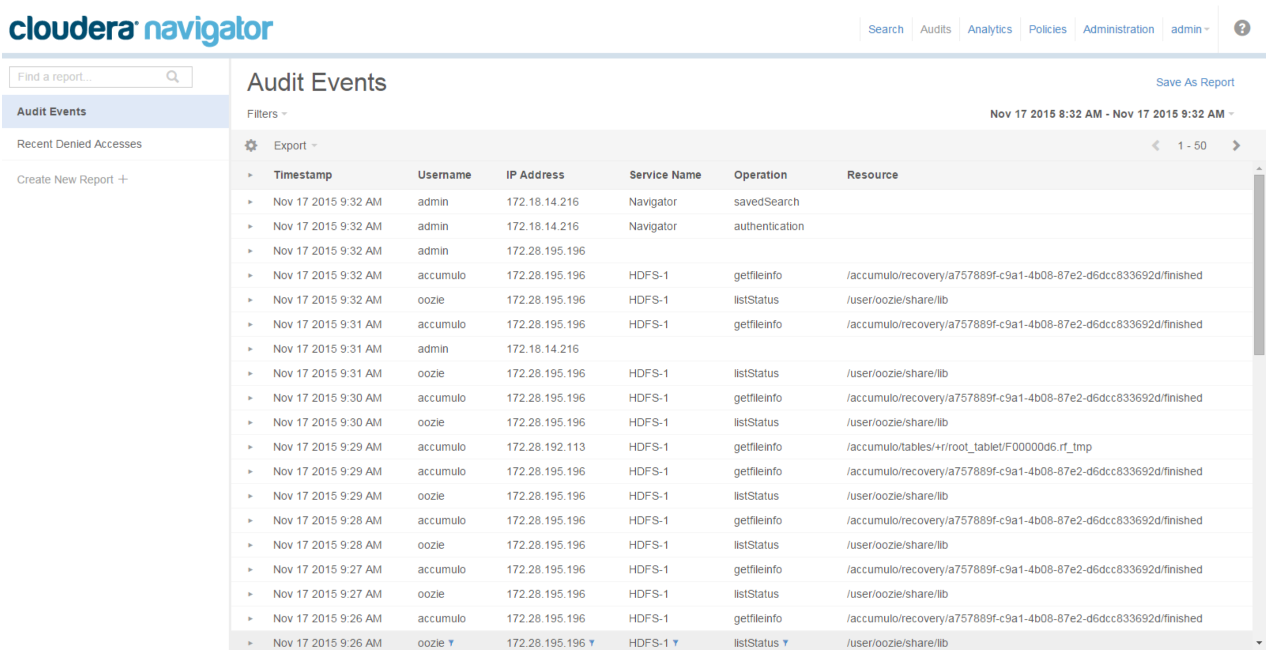
Auditing

While Hadoop has historically lacked centralized cross-component audit capabilities, products such as Cloudera Navigator add secured, real-time audit components to key data and access frameworks. Using Cloudera Navigator, administrators can configure, collect, and view audit events, to understand who accessed what data and how.

Cloudera Navigator also lets administrators generate reports that list the HDFS access permissions granted to groups. Cloudera Navigator tracks access permissions and actual accesses to all entities in HDFS, Hive, HBase, Impala, Sentry, and Solr, and the Cloudera Navigator Metadata Server itself to help answer questions such as - who has access to which entities, which entities were accessed by a user, when was an entity accessed and by whom, what entities were accessed using a service, and which device was used to access. Cloudera Navigator auditing supports tracking access to:

- HDFS entities accessed by HDFS, Hive, HBase, Impala, and Solr services
- HBase and Impala
- Hive metadata
- Sentry
- Solr
- Cloudera Navigator Metadata Server

Data collected from these services also provides visibility into usage patterns for users, ability to see point-in-time permissions and how they have changed (leveraging Sentry), and review and verify HDFS permissions. Cloudera Navigator also provides out-of-the-box integration with leading enterprise metadata, lineage, and SIEM applications.



Metadata Management

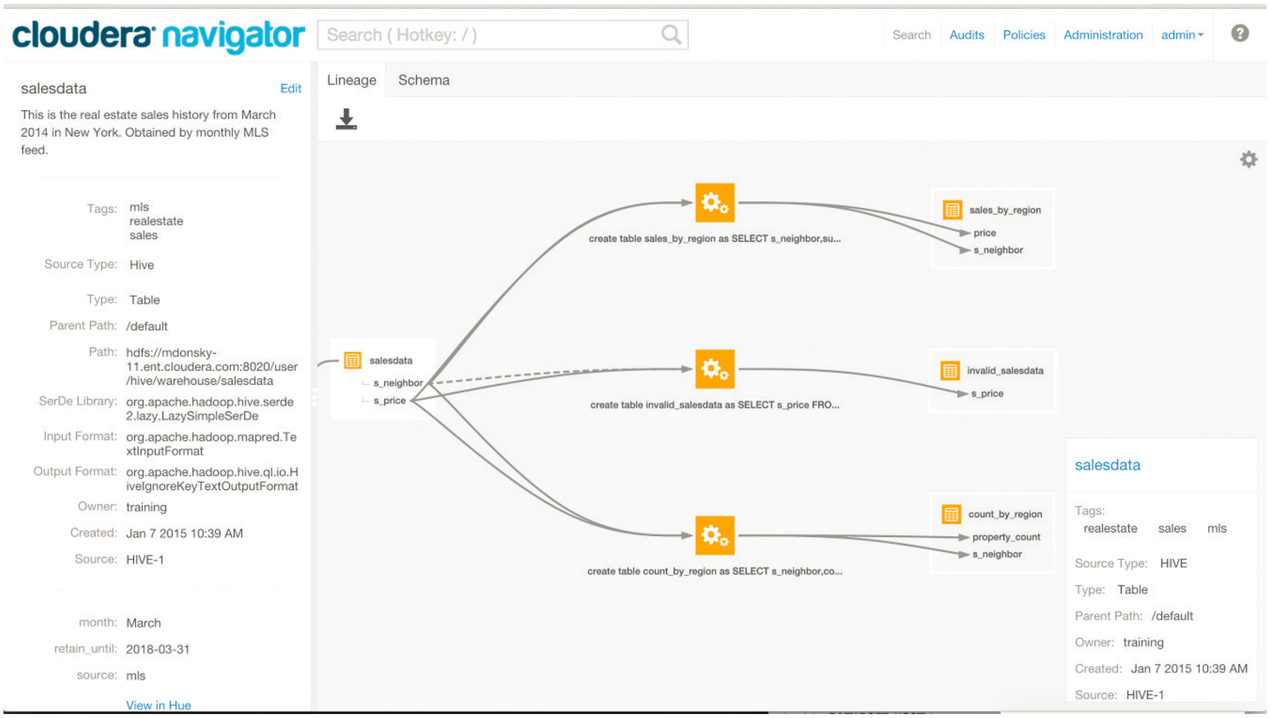
Cloudera Navigator features complete metadata storage and supports data discovery. It consolidates technical metadata for all cluster data and enables automatic tagging of data based on the external sources entering the cluster. The consolidated metadata store is searchable through the Cloudera Navigator console, a web-based unified interface.

In addition, Cloudera Navigator supports user-defined metadata that can be applied to files, tables, and individual columns, to identify data assets for business context. The result is that data stewards can devise appropriate classification schemes for specific business purposes and data is more easily discovered and located by users.

Furthermore, policies can be used to automatically classify and applying metadata to cluster data based on arrival, scheduled interval, or other trigger.

Lineage

Cloudera Navigator lineage is a visualization tool for tracing data and its transformations from upstream to downstream through the cluster. Lineage can show the transforms that produced upstream data sources and the effect the data has on downstream artifacts, to the column level. Cloudera Navigator tracks lineage of HDFS files, datasets, and directories, Hive tables and columns, MapReduce and YARN jobs, Hive queries, Impala queries, Pig scripts, Oozie workflows, Spark jobs, and Sqoop jobs.



Integration within the Enterprise

Monitoring and reporting are only part a subset of enterprise auditing infrastructure capabilities. Enterprise policies may mandate that audit data route be able to integrate with existing enterprise SIEM (security information and event management) applications and other tools. Toward that end, Cloudera Navigator can deliver or export audit data through several mechanisms:

- Using `syslog` as a mediator between raw-event stream generated by Hadoop cluster and the SIEM tools.
- Using a REST API for custom enterprise tools.
- Exporting data to CSV or other text file.

Auditing and Components

The table below details auditing capabilities of Cloudera Manager and CDH components.

Component	Auditing Capabilities
HDFS	Events captured by Cloudera Navigator (including security events)
MapReduce	Inferred through HDFS
YARN	Inferred through HDFS
Accumulo	Log Files - Partial inclusion of security events; does not include non-bulk writes
Flume	Log Files
HBase	Audit events captured by Cloudera Navigator (including security events)
HiveServer2	Audit events captured by Cloudera Navigator

Component	Auditing Capabilities
Hue	Inferred through underlying components
Impala	Audit events captured by Cloudera Navigator
Oozie	Log Files
Pig	Inferred through HDFS
Search	Log Files
Sentry	Audit events captured by Cloudera Navigator
Spark	Inferred through HDFS
Sqoop	Log Files
Sqoop2	Log Files (including security events)
ZooKeeper	Log Files
Cloudera Manager	Audit events captured by Cloudera Navigator (partial capture of security events)
Cloudera Navigator	Audit events captured by Cloudera Navigator itself
Backup and Disaster Recovery	None

Security Events

Security events include a machine readable log of the following activities:

- User data read
- User data written
- Permission changes
- Configuration changes
- Login attempts
- Escalation of privileges
- Session Tracking
- Key Operations (Key Trustee)

Authentication

Authentication is a process that requires users and services to prove their identity when trying to access a system resource. Organizations typically manage user identity and authentication through various time-tested technologies, including Lightweight Directory Access Protocol (LDAP) for identity, directory, and other services, such as group management, and Kerberos for authentication.

Cloudera clusters support integration with both of these technologies. For example, organizations with existing LDAP directory services such as Active Directory (included in Microsoft Windows Server as part of its suite of Active Directory Services) can leverage the organization's existing user accounts and group listings instead of creating new accounts throughout the cluster. Using an external system such as Active Directory or OpenLDAP is required to support the user role authorization mechanism implemented in Cloudera Navigator.

For authentication, Cloudera supports integration with MIT Kerberos and with Active Directory, which includes Kerberos implementation for authentication. Kerberos provides ***strong authentication***, *strong* meaning that cryptographic mechanisms—rather than passwords alone—are used in the exchange between requesting process and service during the authentication process.

These systems are not mutually exclusive. For example, Microsoft Active Directory is an LDAP directory service that also provides Kerberos authentication services, and Kerberos credentials can be stored and managed in an LDAP directory service. Cloudera Manager Server, CDH nodes, and Cloudera Enterprise components, such as Cloudera Navigator, Apache Hive, Hue, and Impala, can all make use of Kerberos authentication.

> **Note:** Cloudera does not provide a Kerberos implementation. Both MIT Kerberos and Microsoft Server Active Directory service are supported.

Configuring the cluster to use Kerberos requires having administrator privileges—and access to—the Kerberos server Key Distribution Center (KDC). The process may require debugging issues between the Cloudera Manager cluster and the KDC.

> **Note:** Integrating clusters to use Microsoft Active Directory as a KDC requires the Windows registry setting for `AllowTgtSessionKey` to be disabled (set to 0). If this registry key has already been enabled, users and credentials are not created—despite the "Successful" message at the end of the configuration/integration process. Before configuring Active Directory for use as KDC, check the value of `AllowTgtSessionKey` on the Active Directory instance and reset to 0 if necessary. See Registry Key to Allow Session Keys to Be Sent in Kerberos Ticket-Granting-Ticket at Microsoft for details.

Configuring Authentication in Cloudera Manager

Cloudera clusters can be configured to use Kerberos for authentication by following a manual configuration process or by using the configuration wizard available from the Cloudera Manager Admin Console. Cloudera recommends using the wizard because it automates many of the configuration and deployment tasks.

> **Important:** Cloudera recommends configuring clusters to use Kerberos authentication after the cluster has been configured for TLS/SSL. See How to Configure TLS Encryption for Cloudera Manager for details.

Cloudera Manager Kerberos Wizard Overview

The Cloudera Manager configuration wizard initiates an information-gathering process through the Cloudera Manager Admin Console that requires entry of very specific details about the Kerberos instance. Specifically, the Key Distribution Center or KDC, is the Kerberos service that disseminates keys after receiving valid tickets from a requesting user or process. Using the wizard requires a working KDC, either an MIT KDC or an Active Directory KDC. For configuration ease, the KDC should be set up and working prior to starting the wizard. Administrator-level privileges to the Kerberos instance are required to complete the prompts of the wizard.

Given the information provided to the wizard entry screens, the configuration wizard does the following:

- Configures the necessary properties in all configuration files—`core-site.xml`, `hdfs-site.xml`, `mapred-site.xml`, and `taskcontroller.cfg`—to identify Kerberos as the authentication mechanism for the cluster
- Configures the necessary properties in the `oozie-site.xml` and `hue.ini` files for Oozie and Hue for Kerberos authentication
- Creates principal and keytab files for core system users, such as `hdfs` and `mapred`, and for CDH services
- Distributes the principal and keytab files to each host in the cluster
- Creates keytab files for `oozie` and hue users and deploys to the appropriate hosts that support these client-focused services
- Distributes a configured `krb5.conf` to all nodes in the cluster
- Stops all services
- Deploys client configurations
- Restarts all services throughout the cluster
- Creates keytab files for core system users, such as `hdfs` and `mapred`, and d

Keytab file for...	Principals
`hdfs`	`hdfs`, `host`
`mapred`	`mapred`, `host`
`oozie`	`oozie`, `HTTP`
`hue`	`hue`

The `host` principal is the same in both `hdfs` and `mapred` keytab files.

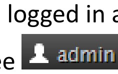
After making the configuration changes and deploying the principals, keytabs, and configuration files to the appropriate nodes in the cluster, Cloudera Manager starts all NameNode, DataNode, Secondary NameNode, JobTracker, TaskTracker, Oozie Server, and Hue roles to stand up the cluster.

- To use the Cloudera Manager wizard, see Enabling Kerberos Authentication Using the Wizard on page 50.
- For command-line configuration, see Enabling Kerberos Authentication Without the Wizard on page 67.

Cloudera Manager User Accounts

Minimum Required Role: User Administrator (also provided by **Full Administrator**)

Access to Cloudera Manager features is controlled by user accounts. A user account identifies how a user is authenticated and determines what privileges are granted to the user.

When you are logged in to the Cloudera Manager Admin Console, the username you are logged in as is located at the far right of the top navigation bar—for example, if you are logged in as *admin* you will see 

.

A user with the User Administrator or Full Administrator role manages user accounts through the **Administration** > **Users** page. View active user sessions on the **User Sessions** tab.

User Authentication

Cloudera Manager provides several mechanisms for authenticating users. You can configure Cloudera Manager to authenticate users against the Cloudera Manager database or against an external authentication service. The external authentication service can be an LDAP server (Active Directory or an OpenLDAP compatible directory), or you can specify another external service. Cloudera Manager also supports using the Security Assertion Markup Language (SAML) to enable single sign-on.

If you are using LDAP or another external service, you can configure Cloudera Manager so that it can use both methods of authentication (internal database and external service), and you can determine the order in which it performs these searches. If you select an external authentication mechanism, Full Administrator users can always authenticate against the Cloudera Manager database. This prevents locking everyone out if the authentication settings are misconfigured, such as with a bad LDAP URL.

With external authentication, you can restrict login access to members of specific groups, and can specify groups whose members are automatically given Full Administrator access to Cloudera Manager.

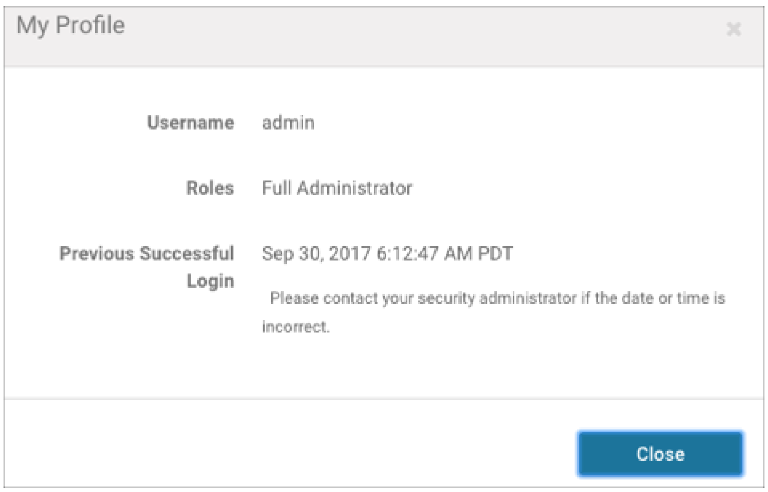
Users accounts in the Cloudera Manager database page show **Cloudera Manager** in the User Type column. User accounts in an LDAP directory or other external authentication mechanism show **External** in the User Type column.

User Roles

User accounts include the user's role, which determines the Cloudera Manager features visible to the user and the actions the user can perform. All tasks in the Cloudera Manager documentation indicate which role is required to perform the task. For more information about user roles, see Cloudera Manager User Roles on page 339.

Determining the Role of the Currently Logged in User

1. Click the logged-in username at the far right of the top navigation bar.
2. Select **My Profile**. The role displays. For example:



Changing the Logged-In Internal User Password

1. Click the logged-in username at the far right of the top navigation bar and select **Change Password**.
2. Enter the current password and a new password twice, and then click **OK**.

Adding an Internal User Account

1. Select **Administration** > **Users**.
2. Click the **Add User** button.
3. Enter a username and password.
4. In the Role drop-down menu, select a role for the new user.
5. Click **Add**.

Assigning User Roles

1. Select **Administration** > **Users**.
2. Check the checkbox next to one or more usernames.
3. Select **Actions for Selected** > **Assign User Roles**.
4. In the drop-down menu, select the role.
5. Click the **Assign Role** button.

Changing an Internal User Account Password

1. Select **Administration** > **Users**.
2. Click the **Change Password** button next to a username with User Type **Cloudera Manager**.
3. Type the new password and repeat it to confirm.
4. Click the **Update** button to make the change.

Deleting Internal User Accounts

1. Select **Administration** > **Users**.
2. Check the checkbox next to one or more usernames with User Type **Cloudera Manager**.
3. Select **Actions for Selected** > **Delete**.
4. Click the **OK** button. (There is no confirmation of the action.)

Viewing User Sessions

1. Select **Administration** > **Users**.
2. Click the tab **User Sessions**.

Configuring External Authentication for Cloudera Manager

Minimum Required Role: [User Administrator] (also provided by **Full Administrator**)

> **Important:** This feature is available only with a Cloudera Enterprise license. It is not available in Cloudera Express. For information on Cloudera Enterprise licenses, see [Managing Licenses].

Cloudera Manager supports user authentication against an internal database and against an external service. The following sections describe how to configure the supported external services.

Configuring Authentication Using Active Directory

1. Log in to Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Select **External Authentication** for the **Category** filter to display the settings.
4. For **Authentication Backend Order**, select the order in which Cloudera Manager should look up authentication credentials for login attempts.
5. For **External Authentication Type**, select **Active Directory**.
6. For **External Authentication Type**, select **Active Directory**.
7. In the **LDAP URL** property, provide the URL of the Active Directory server.
8. In the **Active Directory Domain** property, provide the domain to authenticate against.

   LDAP URL and **Active Directory** are the only settings required to allow anyone in Active Directory to log in to Cloudera Manager.

   For example, if you set LDAP URL to `ldap://adserver.example.com` and the Active Directory Domain to `ADREALM.EXAMPLE.COM`, users can log into Cloudera Manager using just their username, such as `sampleuser`. They no longer require the complete string: `sampleuser@ADREALM.EXAMPLE.COM`.

9. In the **LDAP User Groups** property, optionally provide a comma-separated list of case-sensitive LDAP group names. If this list is provided, only users who are members of one or more of the groups in the list will be allowed to log into Cloudera Manager. If this property is left empty, all authenticated LDAP users will be able to log into Cloudera Manager. For example, if there is a group called `CN=ClouderaManagerUsers,OU=Groups,DC=corp,DC=com`, add the group name `ClouderaManagerUsers` to the **LDAP User Groups** list to allow members of that group to log in to Cloudera Manager.
10. To automatically assign a [role] to users when they log in, provide a comma-separated list of LDAP group names in the following properties:

    - LDAP Full Administrator Groups

- LDAP User Administrator Groups
- LDAP Cluster Administrator Groups
- LDAP BDR Administrator Groups
- LDAP Configurator Groups
- LDAP Key Administrator Groups
- LDAP Navigator Administrator Groups
- LDAP Operator Groups
- LDAP Limited Operator Groups
- LDAP Auditor Groups

If you specify groups in these properties, users must also be a member of at least one of the groups specified in the **LDAP User Groups** property or they will not be allowed to log in. If these properties are left empty, users will be assigned to the Read-Only role and any other role assignment must be performed manually by an Administrator.

> **Note:** Users added to LDAP groups are not automatically assigned user roles from the internal Cloudera Manager database so they are granted Read-Only access.

Configuring Authentication Using an LDAP-compliant Identity Service

An LDAP-compliant identity/directory service, such as OpenLDAP, provides different options for enabling Cloudera Manager to look-up user accounts and groups in the directory:

- Use a single Distinguished Name (DN) as a base and provide a pattern (**Distinguished Name Pattern**) for matching user names in the directory, or
- Search filter options let you search for a particular user based on somewhat broader search criteria – for example Cloudera Manager users could be members of different groups or organizational units (OUs), so a single pattern does not find all those users. Search filter options also let you find all the groups to which a user belongs, to help determine if that user should have login or admin access.

1. Log in to Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Select **External Authentication** for the **Category** filter to display the settings.
4. For **Authentication Backend Order**, select the order in which Cloudera Manager should look up authentication credentials for login attempts.
5. For **External Authentication Type**, select **LDAP**.
6. In the **LDAP URL** property, provide the URL of the LDAP server and (optionally) the base Distinguished Name (DN) (the search base) as part of the URL — for example `ldap://ldap-server.corp.com/dc=corp,dc=com`.
7. If your server does not allow anonymous binding, provide the user DN and password to be used to bind to the directory. These are the **LDAP Bind User Distinguished Name** and **LDAP Bind Password** properties. By default, Cloudera Manager assumes anonymous binding.
8. Use one of the following methods to search for users and groups:

    - You can search using User or Group search filters, using the **LDAP User Search Base**, **LDAP User Search Filter**, **LDAP Group Search Base** and **LDAP Group Search Filter** settings. These allow you to combine a base DN with a search filter to allow a greater range of search targets.

      For example, if you want to authenticate users who may be in one of multiple OUs, the search filter mechanism will allow this. You can specify the User Search Base DN as `dc=corp,dc=com` and the user search filter as `uid={0}`. Then Cloudera Manager will search for the user anywhere in the tree starting from the Base DN. Suppose you have two OUs—`ou=Engineering` and `ou=Operations`—Cloudera Manager will find User "foo" if it exists in either of these OUs, that is, `uid=foo,ou=Engineering,dc=corp,dc=com` or `uid=foo,ou=Operations,dc=corp,dc=com`.

      You can use a user search filter along with a DN pattern, so that the search filter provides a fallback if the DN pattern search fails.

The Groups filters let you search to determine if a DN or username is a member of a target group. In this case, the filter you provide can be something like `member={0}` where `{0}` will be replaced with the **DN** of the user you are authenticating. For a filter requiring the username, `{1}` may be used, as `memberUid={1}`. This will return a list of groups the user belongs to, which will be compared to the list in the group properties discussed in step 8 of Configuring Authentication Using Active Directory on page 43.

OR

- Alternatively, specify a single base Distinguished Name (DN) and then provide a "Distinguished Name Pattern" in the **LDAP Distinguished Name Pattern** property.

  Use `{0}` in the pattern to indicate where the username should go. For example, to search for a distinguished name where the `uid` attribute is the username, you might provide a pattern similar to `uid={0},ou=People,dc=corp,dc=com`. Cloudera Manager substitutes the name provided at login into this pattern and performs a search for that specific user. So if a user provides the username "foo" at the Cloudera Manager login page, Cloudera Manager will search for the DN `uid=foo,ou=People,dc=corp,dc=com`.

  If you provided a base DN along with the URL, the pattern only needs to specify the rest of the DN pattern. For example, if the URL you provide is `ldap://ldap-server.corp.com/dc=corp,dc=com`, and the pattern is `uid={0},ou=People`, then the search DN will be `uid=foo,ou=People,dc=corp,dc=com`.

9. Restart the Cloudera Manager Server.

Configuring Cloudera Manager to Use LDAPS

If the LDAP server certificate has been signed by a trusted Certificate Authority, steps 1 and 2 below may not be necessary.

1. Copy the CA certificate file to the Cloudera Manager Server host.
2. Import the CA certificate(s) from the CA certificate file to the local truststore. The default truststore is located in the `$JAVA_HOME/jre/lib/security/cacerts` file. This contains the default CA information shipped with the JDK. Create an alternate default file called `jssecacerts` in the same location as the `cacerts` file. You can now safely append CA certificates for any private or public CAs not present in the default `cacerts` file, while keeping the original file intact.

   For our example, we will follow this recommendation by copying the default `cacerts` file into the new `jssecacerts` file, and then importing the CA certificate to this alternate truststore.

```
$ cp $JAVA_HOME/jre/lib/security/cacerts \
$JAVA_HOME/jre/lib/security/jssecacerts
```

```
$ /usr/java/latest/bin/keytool -import -alias nt_domain_name \
-keystore /usr/java/latest/jre/lib/security/jssecacerts -file path_to_CA_cert
```

> **Note:** The default password for the `cacerts` store is `changeit`. The `-alias` does not always need to be the domain name.

   Alternatively, you can use the Java options: `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword`. Open the `/etc/default/cloudera-scm-server` file and add the following options:

```
export CMF_JAVA_OPTS="-Xmx2G -XX:MaxPermSize=256m -XX:+HeapDumpOnOutOfMemoryError /

-XX:HeapDumpPath=/tmp
-Djavax.net.ssl.trustStore=/usr/java/default/jre/lib/security/jssecacerts /
-Djavax.net.ssl.trustStorePassword=changeit"
```

3. Configure the **LDAP URL** property to use `ldaps://ldap_server` instead of `ldap://ldap_server`.
4. Restart the Cloudera Manager Server.

Configuring Authentication Using an External Program

Cloudera Manager can use a custom external authentication program,. Typically, this may be a custom script that interacts with a custom authentication service. Cloudera Manager will call the external program with the username as the first command line argument. The password is passed over `stdin`. Cloudera Manager assumes the program will return the following exit codes identifying the user role for a successful authentication:

- 0 - Read-Only
- 1 - Full Administrator
- 2 - Limited Operator
- 3 - Operator
- 4 - Configurator
- 5 - Cluster Administrator
- 6 - BDR Administrator
- 7 - Navigator Administrator
- 8 - User Administrator
- 9 - Auditor
- 10 - Key Administrator

and a negative value is returned for a failure to authenticate.

To configure authentication using an external program:

1. Log in to Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Select **External Authentication** for the **Category** filter to display the settings.
4. For **External Authentication Type**, select **External Program**.
5. Provide a path to the external program in the **External Authentication Program Path** property.

Configuring Authentication Using SAML

Cloudera Manager supports the Security Assertion Markup Language (SAML), an XML-based open standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider (IDP) and a service provider (SP). The SAML specification defines three roles: the principal (typically a user), the IDP, and the SP. In the use case addressed by SAML, the principal (user agent) requests a service from the service provider. The service provider requests and obtains an identity assertion from the IDP. On the basis of this assertion, the SP can make an access control decision—in other words it can decide whether to perform some service for the connected principal.

The primary SAML use case is called web browser single sign-on (SSO). A user wielding a user agent (usually a web browser) requests a web resource protected by a SAML SP. The SP, wanting to know the identity of the requesting user, issues an authentication request to a SAML IDP through the user agent. In the context of this terminology, Cloudera Manager operates as a SP. This topic discusses the Cloudera Manager part of the configuration process; it assumes that you are familiar with SAML and SAML configuration in a general sense, and that you have a functioning IDP already deployed.

Note:

- Cloudera Manager supports both SP- and IDP-initiated SSO.
- The logout action in Cloudera Manager will send a single-logout request to the IDP.
- SAML authentication has been tested with specific configurations of SiteMinder and Shibboleth. While SAML is a standard, there is a great deal of variability in configuration between different IDP products, so it is possible that other IDP implementations, or other configurations of SiteMinder and Shibboleth, may not interoperate with Cloudera Manager.
- To bypass SSO if SAML configuration is incorrect or not working, you can login using a Cloudera Manager local account using the URL: `http://`*`cm_host`*`:7180/cmf/localLogin`

Setting up Cloudera Manager to use SAML requires the following steps.

Preparing Files

You will need to prepare the following files and information, and provide these to Cloudera Manager:

- A Java keystore containing a private key for Cloudera Manager to use to sign/encrypt SAML messages. For guidance on creating Java keystores, see Understanding Keystores and Truststores on page 181.
- The SAML metadata XML file from your IDP. This file must contain the public certificates needed to verify the sign/encrypt key used by your IDP per the SAML Metadata Interoperability Profile. For example, if you are using the Shibboleth IdP, the metadata file is available at: `https://<IdPHOST>:8080/idp/shibboleth`.

> **Note:** For guidance on how to obtain the metadata XML file from your IDP, either contact your IDP administrator or consult the documentation for the version of the IDP you are using.

- The entity ID that should be used to identify the Cloudera Manager instance
- How the user ID is passed in the SAML authentication response:
  - As an attribute. If so, what identifier is used.
  - As the NameID.
- The method by which the Cloudera Manager role will be established:
  - From an attribute in the authentication response:
    - What identifier will be used for the attribute
    - What values will be passed to indicate each role
  - From an external script that will be called for each use:
    - The script takes user ID as $1
    - The script sets an exit code to reflect successful authentication of the assigned role:
      - 0 - Full Administrator
      - 1 - Read-Only
      - 2 - Limited Operator
      - 3 - Operator
      - 4 - Configurator
      - 5 - Cluster Administrator
      - 6 - BDR Administrator
      - 7 - Navigator Administrator
      - 8 - User Administrator
      - 9 - Auditor
      - 10 - Key Administrator

      and a negative value is returned for a failure to authenticate.

Configuring Cloudera Manager

1. Log in to Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Select **External Authentication** for the **Category** filter to display the settings.
4. Set the **External Authentication Type** property to **SAML** (the Authentication Backend Order property is ignored for SAML).
5. Set the **Path to SAML IDP Metadata File** property to point to the IDP metadata file.
6. Set the **Path to SAML Keystore File** property to point to the Java keystore prepared earlier.
7. In the **SAML Keystore Password** property, set the keystore password.

8. In the **Alias of SAML Sign/Encrypt Private Key** property, set the alias used to identify the private key for Cloudera Manager to use.
9. In the **SAML Sign/Encrypt Private Key Password** property, set the private key password.
10. Set the **SAML Entity ID** property if:

    - There is more than one Cloudera Manager instance being used with the same IDP (each instance needs a different entity ID).
    - Entity IDs are assigned by organizational policy.

11. In the **Source of User ID in SAML Response** property, set whether the user ID will be obtained from an attribute or the NameID.

    If an attribute will be used, set the attribute name in the **SAML attribute identifier for user ID** property. The default value is the normal OID used for user IDs and so may not need to be changed.

12. In the **SAML Role assignment mechanism** property, set whether the role assignment will be done from an attribute or an external script.

    - If an attribute will be used:

        – In the **SAML attribute identifier for user role** property, set the attribute name if necessary. The default value is the normal OID used for OrganizationalUnits and so may not need to be changed.
        – In the **SAML Attribute Values for Roles** property, set which attribute values will be used to indicate the user role.

    - If an external script will be used, set the path to that script in the **Path to SAML Role Assignment Script** property. Make sure that the script is executable (an executable binary is fine - it doesn't need to be a shell script).

13. Save the changes. Cloudera Manager will run a set of validations that ensure it can find the metadata XML and the keystore, and that the passwords are correct. If you see a validation error, correct the problem before proceeding.
14. Restart the Cloudera Manager Server.

Configuring the IDP

After the Cloudera Manager Server is restarted, it will attempt to redirect to the IDP login page instead of showing the normal CM page. This may or may not succeed, depending on how the IDP is configured. In either case, the IDP will need to be configured to recognize CM before authentication will actually succeed. The details of this process are specific to each IDP implementation - refer to your IDP documentation for details. If you are using the Shibboleth IdP, information on configuring the IdP to communicate with a Service Provider is available here.

1. Download the Cloudera Manager's SAML metadata XML file from `http://hostname:7180/saml/metadata`.
2. Inspect the metadata file and ensure that any URLs contained in the file can be resolved by users' web browsers. The IDP will redirect web browsers to these URLs at various points in the process. If the browser cannot resolve them, authentication will fail. If the URLs are incorrect, you can manually fix the XML file or set the Entity Base URL in the CM configuration to the right value, and then re-download the file.
3. Provide this metadata file to your IDP using whatever mechanism your IDP provides.
4. Ensure that the IDP has access to whatever public certificates are necessary to validate the private key that was provided to Cloudera Manager earlier.
5. Ensure that the IDP is configured to provide the User ID and Role using the attribute names that Cloudera Manager was configured to expect, if relevant.
6. Ensure the changes to the IDP configuration have taken effect (a restart may be necessary).

Verifying Authentication and Authorization

1. Return to the Cloudera Manager Admin Console and refresh the login page.
2. Attempt to log in with credentials for a user that is entitled. The authentication should complete and you should see the **Home** > **Status** tab.
3. If authentication fails, you will see an IDP provided error message. Cloudera Manager is not involved in this part of the process, and you must ensure the IDP is working correctly to complete the authentication.

4. If authentication succeeds but the user is not authorized to use Cloudera Manager, they will be taken to an error page by Cloudera Manager that explains the situation. If an user who should be authorized sees this error, then you will need to verify their role configuration, and ensure that it is being properly communicated to Cloudera Manager, whether by attribute or external script. The Cloudera Manager log will provide details on failures to establish a user's role. If any errors occur during role mapping, Cloudera Manager will assume the user is unauthorized.

Kerberos Concepts - Principals, Keytabs and Delegation Tokens

This section describes how Hadoop uses Kerberos principals and keytabs for user authentication. It also briefly describes how Hadoop uses delegation tokens to authenticate jobs at execution time, to avoid overwhelming the KDC with authentication requests for each job.

Kerberos Principals

A user in Kerberos is called a ***principal***, which is made up of three distinct components: the primary, instance, and realm. A ***Kerberos principal*** is used in a Kerberos-secured system to represent a unique identity. The first component of the principal is called the ***primary***, or sometimes the user component. The primary component is an arbitrary string and may be the operating system username of the user or the name of a service. The primary component is followed by an optional section called the ***instance***, which is used to create principals that are used by users in special roles or to define the host on which a service runs, for example. An instance, if it exists, is separated from the primary by a slash and then the content is used to disambiguate multiple principals for a single user or service. The final component of the principal is the realm. The ***realm*** is similar to a domain in DNS in that it logically defines a related group of objects, although rather than hostnames as in DNS, the Kerberos realm defines a group of principals . Each realm can have its own settings including the location of the KDC on the network and supported encryption algorithms. Large organizations commonly create distinct realms to delegate administration of a realm to a group within the enterprise. Realms, by convention, are written in uppercase characters.

Kerberos assigns tickets to Kerberos principals to enable them to access Kerberos-secured Hadoop services. For the Hadoop daemon principals, the principal names should be of the format `service/fully.qualified.domain.name@YOUR-REALM.COM`. Here, *service* in the `service/fully.qualified.domain.name@YOUR-REALM.COM` principal refers to the username of an existing Unix account that is used by Hadoop daemons, such as `hdfs` or `mapred`.

Human users who want to access the Hadoop cluster also need to have Kerberos principals of the format, `username@YOUR-REALM.COM`; in this case, `username` refers to the username of the user's Unix account, such as `joe` or `jane`. Single-component principal names (such as `joe@YOUR-REALM.COM`) are typical for client user accounts. Hadoop does not support more than two-component principal names.

Kerberos Keytabs

A ***keytab*** is a file containing pairs of Kerberos principals and an encrypted copy of that principal's key. A keytab file for a Hadoop daemon is unique to each host since the principal names include the hostname. This file is used to authenticate a principal on a host to Kerberos without human interaction or storing a password in a plain text file. Because having access to the keytab file for a principal allows one to act as that principal, access to the keytab files should be tightly secured. They should be readable by a minimal set of users, should be stored on local disk, and should not be included in host backups, unless access to those backups is as secure as access to the local host.

Delegation Tokens

Users in a Hadoop cluster authenticate themselves to the NameNode using their Kerberos credentials. However, once the user is authenticated, each job subsequently submitted must also be checked to ensure it comes from an authenticated user. Since there could be a time gap between a job being submitted and the job being executed, during which the user could have logged off, user credentials are passed to the NameNode using delegation tokens that can be used for authentication in the future.

Delegation tokens are a secret key shared with the NameNode, that can be used to impersonate a user to get a job executed. While these tokens can be renewed, new tokens can only be obtained by clients authenticating to the NameNode using Kerberos credentials. By default, delegation tokens are only valid for a day. However, since jobs can last longer than a day, each token specifies a JobTracker as a *renewer* which is allowed to renew the delegation token once a day, until the job completes, or for a maximum period of 7 days. When the job is complete, the JobTracker requests the NameNode to cancel the delegation token.

Token Format

The NameNode uses a random `masterKey` to generate delegation tokens. All active tokens are stored in memory with their expiry date (`maxDate`). Delegation tokens can either expire when the current time exceeds the expiry date, or, they can be canceled by the owner of the token. Expired or canceled tokens are then deleted from memory. The `sequenceNumber` serves as a unique ID for the tokens. The following section describes how the Delegation Token is used for authentication.

```
TokenID = {ownerID, renewerID, issueDate, maxDate, sequenceNumber}
TokenAuthenticator = HMAC-SHA1(masterKey, TokenID)
Delegation Token = {TokenID, TokenAuthenticator}
```

Authentication Process

To begin the authentication process, the client first sends the TokenID to the NameNode. The NameNode uses this TokenID and the `masterKey` to once again generate the corresponding TokenAuthenticator, and consequently, the Delegation Token. If the NameNode finds that the token already exists in memory, and that the current time is less than the expiry date (`maxDate`) of the token, then the token is considered valid. If valid, the client and the NameNode will then authenticate each other by using the TokenAuthenticator that they possess as the secret key, and MD5 as the protocol. Since the client and NameNode do not actually exchange TokenAuthenticators during the process, even if authentication fails, the tokens are not compromised.

Token Renewal

Delegation tokens must be renewed periodically by the designated renewer (`renewerID`). For example, if a JobTracker is the designated renewer, the JobTracker will first authenticate itself to the NameNode. It will then send the token to be authenticated to the NameNode. The NameNode verifies the following information before renewing the token:

- The JobTracker requesting renewal is the same as the one identified in the token by `renewerID`.
- The TokenAuthenticator generated by the NameNode using the `TokenID` and the `masterKey` matches the one previously stored by the NameNode.
- The current time must be less than the time specified by `maxDate`.

If the token renewal request is successful, the NameNode sets the new expiry date to `min(current time+renew period, maxDate)`. If the NameNode was restarted at any time, it will have lost all previous tokens from memory. In this case, the token will be saved to memory once again, this time with a new expiry date. Hence, designated renewers must renew all tokens with the NameNode after a restart, and before relaunching any failed tasks.

A designated renewer can also revive an expired or canceled token as long as the current time does not exceed `maxDate`. The NameNode cannot tell the difference between a token that was canceled, or has expired, and one that was erased from memory due to a restart, since only the `masterKey` persists in memory. The `masterKey` must be updated regularly.

Enabling Kerberos Authentication Using the Wizard

Required Role: **Cluster Administrator** or **Full Administrator**

Cloudera Manager provides a wizard for integrating your organization's Kerberos instance with your cluster to provide authentication services.

Kerberos must already be deployed in your organization and the Kerberos key distribution center (KDC) must be ready to use, with a realm established. For Hue and Oozie, the Kerberos realm must support renewable tickets.

> **Important:** Before integrating Kerberos with your cluster, configure TLS/SSL encryption between Cloudera Manager Server and all Cloudera Manager Agent host systems in the cluster. During the Kerberos integration process, Cloudera Manager Server sends keytab files to the Cloudera Manager Agent hosts, and TLS/SSL encrypts the network communication so these files are protected. See How to Configure TLS Encryption for Cloudera Manager on page 456 for details.

Cloudera Manager clusters can be integrated with MIT Kerberos or with Microsoft Active Directory:

- See MIT Kerberos home and MIT Kerberos 5 Release 1.8.6 documentation for more information about MIT Kerberos.
- See Direct to Active Directory on page 21 and Microsoft Active Directory documentation for more information about using Active Directory as a KDC.

For Active Directory, you must have administrative privileges to the Active Directory instance for initial setup and for on-going management, or you will need to have the help of your AD administrator prior to and during the integration process. For example, administrative access is needed to access the Active Directory KDC, create principals, and troubleshoot Kerberos TGT/TGS-ticket-renewal and take care of any other issues that may arise.

Kerberos client OS-specific packages must be installed on all cluster hosts and client hosts that will authenticate using Kerberos.

OS	Packages Required
RHEL/CentOS 7, **RHEL/CentOS 6**, **RHEL/CentOS 5**	• `openldap-clients` on the Cloudera Manager Server host • `krb5-workstation`, `krb5-libs` on ALL hosts
SLES	• `openldap2-client` on the Cloudera Manager Server host • `krb5-client` on ALL hosts
Ubuntu or Debian	• `ldap-utils` on the Cloudera Manager Server host • `krb5-user` on ALL hosts
Windows	• `krb5-workstation`, `krb5-libs` on ALL hosts

See Before you Begin Using the Wizard on page 54 for more information.

Cloudera supports the Kerberos version that ships with each supported operating system listed in CDH and Cloudera Manager Supported Operating Systems.

Step 1: Install Cloudera Manager and CDH

If you have not already done so, Cloudera strongly recommends that you install and configure the Cloudera Manager Server and Cloudera Manager Agents and CDH to set up a fully-functional CDH cluster *before* you begin doing the following steps to implement Hadoop security features.

Overview of the User Accounts and Groups in CDH and Cloudera Manager to Support Security

User Accounts and Groups in CDH and Cloudera Manager Required to Support Security:

When you install the CDH packages and the Cloudera Manager Agents on your cluster hosts, Cloudera Manager takes some steps to provide system security such as creating the following Unix accounts and setting directory permissions as shown in the following table. These Unix accounts and directory permissions work with the Hadoop Kerberos security requirements.

This User	Runs These Roles
`hdfs`	NameNode, DataNodes, and Secondary Node
`mapred`	JobTracker and TaskTrackers (MR1) and Job History Server (YARN)
`yarn`	ResourceManager and NodeManagers (YARN)

This User	Runs These Roles
oozie	Oozie Server
hue	Hue Server, Beeswax Server, Authorization Manager, and Job Designer

The `hdfs` user also acts as the HDFS superuser.

When you install the Cloudera Manager Server on the server host, a new Unix user account called `cloudera-scm` is created automatically to support security. The Cloudera Manager Server uses this account to create and deploy the host principals and keytabs on your cluster.

Depending on whether you installed CDH and Cloudera Manager at the same time or not, use one of the following sections for information on configuring directory ownerships on cluster hosts:

If you installed CDH and Cloudera Manager at the Same Time

If you have a new installation and you installed CDH and Cloudera Manager at the same time, when you started the Cloudera Manager Agents on your cluster hosts, the Cloudera Manager Agent on each host automatically configured the directory owners shown in the following table to support security. Assuming the owners are configured as shown, the Hadoop daemons can then automatically set the permissions for each of the directories specified by the properties shown below to make sure they are properly restricted. It's critical that the owners are configured exactly as shown below, so do not change them:

Directory Specified in this Property	Owner
`dfs.name.dir`	`hdfs:hadoop`
`dfs.data.dir`	`hdfs:hadoop`
`mapred.local.dir`	`mapred:hadoop`
`mapred.system.dir` in HDFS	`mapred:hadoop`
`yarn.nodemanager.local-dirs`	`yarn:yarn`
`yarn.nodemanager.log-dirs`	`yarn:yarn`
`oozie.service.StoreService.jdbc.url` (if using Derby)	`oozie:oozie`
`[[database]] name`	`hue:hue`
`javax.jdo.option.ConnectionURL`	`hue:hue`

If you Installed and Used CDH Before Installing Cloudera Manager

If you have been using HDFS and running MapReduce jobs in an existing installation of CDH before you installed Cloudera Manager, you must manually configure the owners of the directories shown in the table above. Doing so enables the Hadoop daemons to automatically set the permissions for each of the directories. It's critical that you manually configure the owners exactly as shown above.

Step 2: If You are Using AES-256 Encryption, Install the JCE Policy File

If you are using CentOS or Red Hat Enterprise Linux 5.5 or higher, which use AES-256 encryption by default for tickets, you must install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy File on all cluster and Hadoop user hosts. There are 2 ways to do this:

- In the Cloudera Manager Admin Console, navigate to the **Hosts** page. Both, the **Add New Hosts to Cluster** wizard and the **Re-run Upgrade Wizard** will give you the option to have Cloudera Manager install the JCE Policy file for you.
- You can follow the JCE Policy File installation instructions in the `README.txt` file included in the `jce_policy-x.zip` file.

Alternatively, you can configure Kerberos to not use AES-256 by removing `aes256-cts:normal` from the `supported_enctypes` field of the `kdc.conf` or `krb5.conf` file. Note that after changing the `kdc.conf` file, you'll need to restart both the KDC and the kadmin server for those changes to take affect. You may also need to recreate or change the password of the relevant principals, including potentially the Ticket Granting Ticket principal (krbtgt/REALM@REALM). If AES-256 is still used after all of those steps, it's because the `aes256-cts:normal` setting existed when the Kerberos database was created. To fix this, create a new Kerberos database and then restart both the KDC and the kadmin server.

To verify the type of encryption used in your cluster:

1. **For MIT KDC:** On the local KDC host, type this command in the kadmin.local or kadmin shell to create a test principal:

```
kadmin:  addprinc test
```

   For Active Directory: Create a new AD account with the name, `test`.

2. On a cluster host, type this command to start a Kerberos session as test:

```
$ kinit test
```

3. On a cluster host, type this command to view the encryption type in use:

```
$ klist -e
```

   If AES is being used, output like the following is displayed after you type the `klist` command (note that `AES-256` is included in the output):

```
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: test@Cloudera Manager
Valid starting     Expires            Service principal
05/19/11 13:25:04  05/20/11 13:25:04  krbtgt/Cloudera Manager@Cloudera Manager
    Etype (skey, tkt): AES-256 CTS mode with 96-bit SHA-1 HMAC, AES-256 CTS mode with
96-bit SHA-1 HMAC
```

Step 3: Get or Create a Kerberos Principal for the Cloudera Manager Server

In order to create and deploy the host principals and keytabs on your cluster, the Cloudera Manager Server must have the correct Kerberos principal. Specifically, the Cloudera Manager Server must have a Kerberos principal that has privileges to create other accounts.

To get or create the Kerberos principal for the Cloudera Manager Server, you can do either of the following:

- Ask your Kerberos administrator to create a Kerberos administrator principal for the Cloudera Manager Server.
- Create the Kerberos principal for the Cloudera Manager Server yourself by using the following instructions in this step.

If for some reason, you cannot create a Cloudera Manager administrator principal on your KDC with the privileges to create other principals and keytabs for CDH services, then these will need to be created manually, and then retrieved by Cloudera Manager. See, Using a Custom Kerberos Keytab Retrieval Script on page 65.

Creating the Cloudera Manager Principal

The following instructions illustrate an example of creating the Cloudera Manager Server principal for MIT KDC and Active Directory KDC. (If you are using another version of Kerberos, refer to your Kerberos documentation for instructions.)

If you are using Active Directory:

1. Create an Organizational Unit (OU) in your AD setup where all the principals used by your CDH cluster will reside.
2. Add a new user account to Active Directory, for example, `<username>@YOUR-REALM.COM`. The password for this user should be set to never expire.

3. Use AD's Delegate Control wizard to allow this new user to **Create, Delete and Manage User Accounts**.

If you are using MIT KDC:

Typically, principals with the second component of `admin` in the principal name (for example, `username/admin@YOUR-LOCAL-REALM.com`) have administrator privileges. This is why `admin` is shown in the following example.

> **Note:** If you are running `kadmin` and the Kerberos Key Distribution Center (KDC) on the same host, use `kadmin.local` in the following steps. If the Kerberos KDC is running on a remote host, you must use `kadmin` instead of `kadmin.local`.

In the `kadmin.local` or `kadmin` shell, type the following command to create the Cloudera Manager Server principal, replacing `YOUR-LOCAL-REALM.COM` with the name of your realm:

```
kadmin:  addprinc -pw <Password> cloudera-scm/admin@YOUR-LOCAL-REALM.COM
```

Step 4: Enabling Kerberos Using the Wizard

Minimum Required Role: Full Administrator

To start the Kerberos wizard:

1. Go to the Cloudera Manager Admin Console and click ▼ to the right of the cluster for which you want to enable Kerberos authentication.
2. Select **Enable Kerberos**.

Before you Begin Using the Wizard

The Welcome page lists steps you should have completed before starting the wizard.

- Set up a working KDC. Cloudera Manager supports authentication with MIT KDC and Active Directory.
- Configure the KDC to allow renewable tickets with non-zero ticket lifetimes.

  Active Directory allows renewable tickets with non-zero lifetimes by default. You can verify this by checking **Domain Security Settings** > **Account Policies** > **Kerberos Policy** in Active Directory.

  For MIT KDC, make sure you have the following lines in the `kdc.conf`.

```
max_life = 1d
max_renewable_life = 7d
kdc_tcp_ports = 88
```

- If you are using Active Directory, make sure LDAP over TLS/SSL (LDAPS) is enabled for the Domain Controllers.
- Install the OS-specific packages for your cluster listed in the table:

OS	Packages Required
RHEL/CentOS 7, RHEL/CentOS 6, RHEL/CentOS 5	• `openldap-clients` on the Cloudera Manager Server host • `krb5-workstation`, `krb5-libs` on ALL hosts
SLES	• `openldap2-client` on the Cloudera Manager Server host • `krb5-client` on ALL hosts
Ubuntu or Debian	• `ldap-utils` on the Cloudera Manager Server host • `krb5-user` on ALL hosts
Windows	• `krb5-workstation`, `krb5-libs` on ALL hosts

- Create an account for Cloudera Manager that has the permissions to create other accounts in the KDC. This should have been completed as part of Step 3: Get or Create a Kerberos Principal for the Cloudera Manager Server on page 53.

> **Important:**
>
> If YARN Resource Manager HA has been enabled in a non-secure cluster, before enabling Kerberos you must clear the StateStore znode in ZooKeeper, as follows:
>
> 1. Go to the Cloudera Manager Admin Console home page, click to the right of the YARN service and select **Stop**.
> 2. When you see a **Finished** status, the service has stopped.
> 3. Go to the YARN service and select **Actions** > **Format State Store**.
> 4. When the command completes, click **Close**.

Once you are able to check all the items on this list, click **Continue**.

KDC Information

On this page, select the KDC type you are using, MIT KDC or Active Directory, and complete the fields as applicable to enable Cloudera Manager to generate principals/accounts for the CDH services running on the cluster.

> **Note:**
>
> - If you are using AD and have multiple Domain Controllers behind a Load Balancer, enter the name of the Load Balancer in the **KDC Server Host** field and any *one* of the Domain Controllers in **Active Directory Domain Controller Override**. Hadoop daemons will use the Load Balancer for authentication, but Cloudera Manager will use the override for creating accounts.
> - If you have multiple Domain Controllers (in case of AD) or MIT KDC servers, only enter the name of any *one* of them in the **KDC Server Host** field. Cloudera Manager will use that server only for creating accounts. If you choose to use Cloudera Manager to manage `krb5.conf`, you can specify the rest of the Domain Controllers using Safety Valve as explained below.
> - Make sure the entries for the **Kerberos Encryption Types** field matches what your KDC supports.
> - If you are using an Active Directory KDC, you can configure Active Directory account properties such as `objectClass` and `accountExpires` directly from the Cloudera Manager UI. You can also enable Cloudera Manager to delete existing AD accounts so that new ones can be created when Kerberos credentials are being regenerated. See Managing Active Directory Account Properties on page 63.

Click **Continue** to proceed.

KRB5 Configuration

Manage krb5.conf through Cloudera Manager allows you to choose whether Cloudera Manager should deploy the `krb5.conf` on your cluster or not. If left unchecked, you must ensure that the `krb5.conf` is deployed on all hosts in the cluster, including the Cloudera Manager Server's host.

If you check **Manage krb5.conf through Cloudera Manager**, this page will let you configure the properties that will be emitted in it. In particular, the safety valves on this page can be used to configure cross-realm authentication. More information can be found at Configuring a Cluster-dedicated MIT KDC with Cross-Realm Trust on page 168.

> **Note:** Cloudera Manager is unable to use a non-default realm. You must specify the default realm.

Click **Continue** to proceed.

Import KDC Account Manager Credentials

Enter the username and password for the user that can create principals for CDH cluster in the KDC. This is the user/principal you created in [Step 3: Get or Create a Kerberos Principal for the Cloudera Manager Server](#) on page 53. Cloudera Manager encrypts the username and password into a keytab and uses it as needed to create new principals.

> **Note:** The username entered should have the realm portion in upper-case only as shown in the example in the UI.

Click **Continue** to proceed.

(Optional) Configuring Custom Kerberos Principals

Starting with Cloudera Manager 5.4, you can configure custom service principals for CDH services. Before you begin making configuration changes, see [Configuring a Cluster with Custom Kerberos Principals](#) on page 61 for some additional configuration changes required and limitations.

Configure HDFS DataNode Ports

On this page, specify the privileged ports needed by the DataNode's Transceiver Protocol and the HTTP Web UI in a secure cluster.

Use the checkbox to confirm you are ready to restart the cluster. Click **Continue**.

Enabling Kerberos

This page lets you track the progress made by the wizard as it first stops all services on your cluster, deploys the `krb5.conf`, generates keytabs for other CDH services, deploys client configuration and finally restarts all services. Click **Continue**.

Congratulations

The final page lists the cluster(s) for which Kerberos has been successfully enabled. Click **Finish** to return to the Cloudera Manager Admin Console home page.

Step 5: Create the HDFS Superuser

To be able to create home directories for users, you will need access to the HDFS superuser account. (CDH automatically created the HDFS superuser account on each cluster host during CDH installation.) When you enabled Kerberos for the HDFS service, you lost access to the default HDFS superuser account using `sudo -u hdfs` commands. Cloudera recommends you use a different user account as the superuser, not the default `hdfs` account.

Designating a Non-Default Superuser Group

To designate a different group of superusers instead of using the default `hdfs` account, follow these steps:

1. Go to the Cloudera Manager Admin Console and navigate to the HDFS service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HDFS (Service-Wide)**.
4. Select **Category** > **Security**.
5. Locate the **Superuser Group** property and change the value to the appropriate group name for your environment. For example, *<superuser>*.
6. Click **Save Changes** to commit the changes.
7. Restart the HDFS service.

   To enable your access to the superuser account now that Kerberos is enabled, you must now create a Kerberos principal or an Active Directory user whose first component is *<superuser>*:

If you are using Active Directory

Add a new user account to Active Directory, `<superuser>@YOUR-REALM.COM`. The password for this account should be set to never expire.

If you are using MIT KDC

1. In the `kadmin.local` or `kadmin` shell, type the following command to create a Kerberos principal called *<superuser>*:

```
kadmin:  addprinc <superuser>@YOUR-LOCAL-REALM.COM
```

   This command prompts you to create a password for the *<superuser>* principal. You should use a strong password because having access to this principal provides superuser access to all of the files in HDFS.
2. To run commands as the HDFS superuser, you must obtain Kerberos credentials for the *<superuser>* principal. To do so, run the following command and provide the appropriate password when prompted.

```
$ kinit <superuser>@YOUR-LOCAL-REALM.COM
```

Step 6: Get or Create a Kerberos Principal for Each User Account

Now that Kerberos is configured and enabled on your cluster, you and every other Hadoop user must have a Kerberos principal or keytab to obtain Kerberos credentials to be allowed to access the cluster and use the Hadoop services. In the next step of this procedure, you will need to create your own Kerberos principals to verify that Kerberos security is working on your cluster. If you and the other Hadoop users already have a Kerberos principal or keytab, or if your Kerberos administrator can provide them, you can skip ahead to the next step.

The following instructions explain how to create a Kerberos principal for a user account.

If you are using Active Directory

Add a new AD user account for each new user that should have access to the cluster. You do not need to make any changes to existing user accounts.

If you are using MIT KDC

1. In the `kadmin.local` or `kadmin` shell, use the following command to create user principals by replacing `YOUR-LOCAL-REALM.COM` with the name of your realm, and replacing USERNAME with a username:

```
kadmin:  addprinc USERNAME@YOUR-LOCAL-REALM.COM
```

2. Enter and re-enter a password when prompted.

Step 7: Prepare the Cluster for Each User

Before you and other users can access the cluster, there are a few tasks you must do to prepare the hosts for each user.

1. Make sure all hosts in the cluster have a Linux user account with the same name as the first component of that user's principal name. For example, the Linux account `joe` should exist on every box if the user's principal name is `joe@YOUR-REALM.COM`. You can use LDAP for this step if it is available in your organization.

   > **Note:** Each account must have a user ID that is greater than or equal to 1000. In the `/etc/hadoop/conf/taskcontroller.cfg` file, the default setting for the `banned.users` property is `mapred`, `hdfs`, and `bin` to prevent jobs from being submitted using those user accounts. The default setting for the `min.user.id` property is `1000` to prevent jobs from being submitted with a user ID less than 1000, which are conventionally Unix super users.

2. Create a subdirectory under `/user` on HDFS for each user account (for example, `/user/joe`). Change the owner and group of that directory to be the user.

```
$ hadoop fs -mkdir /user/joe
$ hadoop fs -chown joe /user/joe
```

Note: `sudo -u hdfs` is not included in the commands above. This is because it is not required if Kerberos is enabled on your cluster. You will, however, need to have Kerberos credentials for the HDFS super user to successfully run these commands. For information on gaining access to the HDFS super user account, see Step 13: Create the HDFS Superuser Principal on page 75

Step 8: Verify that Kerberos Security is Working

After you have Kerberos credentials, you can verify that Kerberos security is working on your cluster by trying to run MapReduce jobs. To confirm, try launching a sleep or a pi job from the provided Hadoop examples (`/usr/lib/hadoop/hadoop-examples.jar`).

Note:

This section assumes you have a fully-functional CDH cluster and you have been able to access HDFS and run MapReduce jobs before you followed these instructions to configure and enable Kerberos on your cluster. If you have not already done so, you should at a minimum use the Cloudera Manager Admin Console to generate a client configuration file to enable you to access the cluster. For instructions, see Deploying Client Configuration Files.

To verify that Kerberos security is working:

1. Acquire Kerberos credentials for your user account.

```
$ kinit USERNAME@YOUR-LOCAL-REALM.COM
```

2. Enter a password when prompted.
3. Submit a sample pi calculation as a test MapReduce job. Use the following command if you use a package-based setup for Cloudera Manager:

```
$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi 10 10000
Number of Maps = 10
Samples per Map = 10000
...
Job Finished in 38.572 seconds
Estimated value of Pi is 3.14120000000000000000
```

If you have a parcel-based setup, use the following command instead:

```
$ hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi
 10 10000
Number of Maps = 10
Samples per Map = 10000
...
Job Finished in 30.958 seconds
Estimated value of Pi is 3.14120000000000000000
```

You have now verified that Kerberos security is working on your cluster.

> **Important:**
>
> Running a MapReduce job will fail if you do not have a valid Kerberos ticket in your credentials cache. You can examine the Kerberos tickets currently in your credentials cache by running the `klist` command. You can obtain a ticket by running the `kinit` command and either specifying a keytab file containing credentials, or entering the password for your principal. If you do not have a valid ticket, you will receive an error such as:
>
> ```
> 11/01/04 12:08:12 WARN ipc.Client:
> Exception encountered while connecting to the server :
> javax.security.sasl.SaslException:GSS initiate failed
> [Caused by GSSException: No valid credentials provided (Mechanism level:
>  Failed to find any
> Kerberos tgt)]
> Bad connection to FS. command aborted. exception: Call to
> nn-host/10.0.0.2:8020 failed on local exception:
> java.io.IOException:javax.security.sasl.SaslException: GSS initiate
> failed
> [Caused by GSSException: No valid credentials provided
> (Mechanism level: Failed to find any Kerberos tgt)]
> ```

Step 9: (Optional) Enable Authentication for HTTP Web Consoles for Hadoop Roles

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

Authentication for access to the HDFS, MapReduce, and YARN roles' web consoles can be enabled using a configuration option for the appropriate service. To enable this authentication:

1. From the **Clusters** tab, select the service (HDFS, MapReduce, or YARN) for which you want to enable authentication.
2. Click the **Configuration** tab.
3. Select **Scope** > ***service name*** **Service-Wide**.
4. Select **Category** > **Security**.
5. Type `Enable Kerberos` in the Search box.
6. Select **Enable Kerberos Authentication for HTTP Web-Consoles**.
7. Click **Save Changes** to commit the changes.
8. When the command finishes, restart all roles of that service.

Enabling SPNEGO as an Authentication Backend for Hue

1. In Cloudera Manager, set the authentication backend to `SpnegoDjangoBackend`.
   a. Go to the Cloudera Manager Admin Console. From the **Clusters** tab, select the Hue service.
   b. Click the **Configuration** tab.
   c. Select **Scope** > **Service-Wide**.
   d. Select **Category** > **Security**.
   e. Locate the **Authentication Backend** property and select `desktop.auth.backend.SpnegoDjangoBackend`.
   f. Click **Save Changes**.
2. Restart the Hue service.
3. If you are using an external load balancer, perform the following steps, otherwise skip the remaining steps. Cloudera Manager creates these configuration automatically:
   a. On the host running the Hue Kerberos Ticket Renewer, switch to the KT_RENEWER process directory. For example:

```
cd /var/run/cloudera-scm-agent/process/`ls -lrt /var/run/cloudera-scm-agent/process/
 \
| awk '{print $9}' |grep KT_RENEWER| tail -1`/
```

b. Verify that the Hue keytab includes the HTTP principal.

```
klist -kte ./hue.keytab

Keytab name: FILE:./hue.keytab
KVNO Timestamp Principal
---- ----------------- --------------------------------------------------------
1 03/09/15 20:20:35 hue/host-10-16-8-168.openstacklocal@EXAMPLE.CLOUDERA.COM
(aes128-cts-hmac-sha1-96)
1 03/09/15 20:20:36 HTTP/host-10-16-8-168.openstacklocal@EXAMPLE.CLOUDERA.COM
(aes128-cts-hmac-sha1-96)
```

c. Copy the `hue.keytab` file to `/var/lib/hue` and change ownership to the hue user and group.

```
$ cp ./hue.keytab /var/lib/hue/
$ chown hue:hue /var/lib/hue/hue.keytab
```

d. Go to the Cloudera Manager Admin Console. From the **Clusters** tab, select the Hue service.

e. Click the **Configuration** tab.

f. Select **Scope** > **Service-Wide**.

g. Select **Category** > **Advanced**.

h. Locate the **Hue Service Environment Advanced Configuration Snippet (Safety Valve)** property and add the following line:

```
KRB5_KTNAME=/var/lib/hue/hue.keytab
```

i. Click **Save Changes** to commit the changes.

j. Restart the Hue service.

Enabling Kerberos Authentication for Single User Mode *or* Non-Default Users

The steps described in this topic are only applicable in the following cases:

- You are running the Cloudera Manager in the single user mode. In this case, configure all the services described in the table below.

  OR

- You are running one or more CDH services with non-default users. This means if you have modified the default value for the **System User** property for any service in Cloudera Manager, you must only perform the command (as described below) corresponding to that service, to be able to successfully run jobs with the non-default user.

MapReduce	Configure the `mapred.system.dir` directory to be owned by the `mapred` user. `sudo -u hdfs hadoop fs -chown mapred:hadoop ${mapred.system.dir}` By default, `mapred.system.dir` is `/tmp/mapred/system`.
HBase	Give the `hbase` user ownership of the HBase root directory: `sudo -u hdfs hadoop fs -chown -R hbase ${hbase.rootdir}` By default, `hbase.rootdir` is `/hbase`.
Hive	Give the `hive` user ownership of the `/user/hive` directory. `sudo -u hdfs hadoop fs -chown hive /user/hive`

YARN	For every NodeManager host, for each path in `yarn.nodemanager.local-dirs`, run: `rm -rf ${yarn.nodemanager.local-dirs}/usercache/*` This removes the `/usercache` directory that contains intermediate data stored for previous jobs.

Configuring a Cluster with Custom Kerberos Principals

By default, the Cloudera Manager Kerberos wizard configures CDH services to use the same Kerberos principals as the default process users. For example, the `hdfs` principal for the HDFS service, and the `hive` principal for the Hive service. The advantage to this is that when Kerberos is enabled, no HDFS directory permissions need to be changed for the new principals. However, starting with Cloudera Manager 5.4, you can configure custom service principals for CDH services.

Important Considerations

- Using different Kerberos principals for different services will make it easier to track the HDFS directories being accessed by each service.
- If you are using `ShellBasedUnixGroupsMapping` to obtain user-group mappings, ensure you have the UNIX accounts for the principals present on all hosts of the cluster.

Configuring Directory Permissions

Configure the following HDFS directories to give their corresponding custom service principals `read, write` and `execute` permissions.

Service	HDFS Directory
Accumulo	• HDFS Directory • `/user/`*`principal`*
HBase	HBase Root Directory
Hive	• Hive Warehouse Directory • `/user/`*`principal`*
Impala	`/user/`*`principal`*
MapReduce v1	`/tmp/mapred`
Oozie	Oozie ShareLib Root Directory
Solr	HDFS Data Directory
Spark on YARN	• `/user/`*`principal`* • Spark History Location • Spark Jar Location
Sqoop2	`/user/`*`principal`*

Configuring CDH Services

The following services will require additional settings if you are using custom principals:

- **HDFS** - If you have enabled synchronization of HDFS and Sentry permissions, add the Hive and Impala principals to the **Sentry Authorization Provider Group** property.

  1. Go to the HDFS service.
  2. Click Configuration.
  3. Select **Scope** > **HDFS Service-Wide**.

4. Select **Category** > **Security**.
5. Locate the **Sentry Authorization Provider Group** property and add the custom Hive and Impala principals.
6. Click **Save Changes**.

- **YARN** - The principals used by YARN daemons should be part of hadoop group so that they are allowed to read JobHistory Server data.
- **Impala** - If you are running the Hue service with a custom principal, configure Impala to allow the Hue principal to impersonate other users.

  1. Go to the Impala service.
  2. Click Configuration.
  3. Select **Scope** > **Impala (Service-Wide)**.
  4. Select **Category** > **Policy File-Based Sentry**.
  5. Locate the **Proxy User Configuration** property and add the custom Hue principal.
  6. Click **Save Changes**.

- **Hive** - If the Sentry service is enabled, allow the Kerberos principals used by Hive, Impala, Hue, HDFS and the Service Monitor to bypass Sentry authorization in the Hive metastore.

  1. Go to the Hive service.
  2. Click Configuration.
  3. Select **Scope** > **Impala (Service-Wide)**.
  4. Select **Category** > **Policy File-Based Sentry**.
  5. Locate the **Bypass Sentry Authorization Users** property and add the custom Hive, Impala, Hue and HDFS principals to the list.
  6. Click **Save Changes**.

- **Spark on YARN** - The principal used by the Spark service should be part of the `spark` group.
- **Sentry** - Allow the Hive, Impala, Hue and HDFS principals to connect to the Sentry service.

  1. Go to the Sentry service.
  2. Click Configuration.
  3. Search for the **Allowed Connecting Users** property and add the custom Hive, Impala, Hue and HDFS principals to the list.
  4. Search for the **Admin Groups** property and include the groups to which the Hive, Impala, and Hue principals belong.
  5. Click **Save Changes**.

- **Cloudera Management Service** - Configure the Reports Manager principal and the Navigator principal for HDFS as HDFS superusers.

  1. Go to the Cloudera Management Service.
  2. Click Configuration.
  3. Search for *kerberos*.
  4. Locate the **Reports Manager Kerberos Principal** property and set it to a principal with administrative and superuser privileges on all HDFS services.
  5. Locate the **Navigator Kerberos Principal for HDFS** property and set it to a principal with administrative and superuser privileges on all HDFS services.
  6. Click **Save Changes**.

Incompatibilities

The following features do not work with custom principals:

- Llama must always use the default Kerberos principal `llama`.
- If you are using MapReduce v1, the Activity Monitor and Cloudera Navigator should use the same principal as the Hue service.

- If you are using the Java KeyStore KMS or KeyTrustee KMS with a custom principal, you will need to add the proxy user for the custom principal to the `kms-site.xml` safety valve.

  For example, if you've replaced the default `oozie` principal with `oozieprinc`, add the `hadoop.kms.proxyuser.oozieprinc.groups` and `hadoop.kms.proxyuser.oozieprinc.hosts` properties to the `kms-site.xml` safety valve.

Managing Kerberos Credentials Using Cloudera Manager

Minimum Required Role: Full Administrator

As soon as you enable Hadoop secure authentication for HDFS and MapReduce service instances, Cloudera Manager starts creating the Kerberos principals for each of the role instances. The amount of time this process will take depends on the number of hosts and HDFS and MapReduce role instances on your cluster. The process can take from a few seconds for a small cluster to several minutes for a larger cluster. After the process is completed, you can use the Cloudera Manager Admin Console to view the list of Kerberos principals that Cloudera Manager has created for the cluster. Make sure there are principals for each of the hosts and HDFS and MapReduce role instances on your cluster. If there are no principals after 10 minutes, then there is most likely a problem with the principal creation. See the Troubleshooting Authentication Issues on page 492 section for more information. If necessary, you can use Cloudera Manager to regenerate the principals.

> **Important:**
>
> - Regenerate principals using the Cloudera Manager Admin Console only, and not directly using `kadmin` shell.
> - Do not regenerate the principals for your cluster unless you have made a global configuration change, such as changing the encryption type.
> - If you are using an MIT KDC, be sure to read Configuring a Cluster-dedicated MIT KDC with Cross-Realm Trust on page 168 to avoid making your existing host keytabs invalid.

Managing Active Directory Account Properties

If you are using an Active Directory KDC, Cloudera Manager 5.8 (and higher) will allow you to configure Active Directory accounts and customize the credential regeneration process using the Cloudera Manager Admin Console. You can also use Cloudera Manager to configure the encryption types to be used by your Active Directory account. Once you modify any Active Directory account properties, you must regenerate Kerberos credentials to reflect those changes. The credential regeneration process requires you to delete existing accounts before new ones are created.

By default, Cloudera Manager does *not* delete accounts in Active Directory. Hence, to regenerate Kerberos principals contained in Active Directory, you need to manually delete the existing Active Directory accounts. You can either delete and regenerate *all* existing Active Directory accounts, or only delete those with the userPrincipalName (or login name) that you will later manually select for regeneration. If the accounts haven't already been deleted manually, the regeneration process will throw an error message saying that deletion of accounts is required before you proceed.

Modifying Active Directory Account Properties Using Cloudera Manager

If you are using an Active Directory KDC, you can configure Active Directory account properties such as `objectClass` and `accountExpires` directly from the Cloudera Manager Admin Console. Any changes to these properties will be reflected in the regenerated Kerberos credentials. To configure AD account properties:

1. Go to the Cloudera Manager Admin Console and click the **Administration** tab.
2. Select **Administration** > **Settings**.
3. Click the **Kerberos** category.
4. Locate the **Active Directory Account Properties** and edit as required. By default, the property will be set to:

```
accountExpires=0,objectClass=top,objectClass=person,objectClass=organizationalPerson,objectClass=user
```

5. Locate the **Active Directory Password Properties** and edit the field as needed. By default, the property will be set to:

```
length=12,minLowerCaseLetters=2,minUpperCaseLetters=2,minDigits=2,minSpaces=0,minSpecialChars=0,specialChars=?.!$%^*()-_+=~
```

6. Click **Save Changes** to commit the changes.
7. Regenerate Kerberos credentials with the new properties.

Enabling Credential Regeneration for Active Directory Accounts Using Cloudera Manager

To avoid having to delete accounts manually, use the following steps to set the **Active Directory Delete Accounts on Credential Regeneration** property to allow Cloudera Manager to automatically delete existing Active Directory accounts when new ones are created during regeneration. If this property is left unchecked (which is the default), Cloudera Manager will not be able to regenerate credentials automatically.

1. Go to the Cloudera Manager Admin Console and click the **Administration** tab.
2. Select **Administration** > **Settings**.
3. Click the **Kerberos** category.
4. Locate the **Active Directory Delete Accounts on Credential Regeneration** and check this property.
5. Click **Save Changes** to commit the changes.

Configuring Encryption Types for Active Directory KDC Using Cloudera Manager

Cloudera Manager allows you to configure the encryption types (or `enctype`) used by an Active Directory KDC to protect its data. Cloudera supports the following encryption types:

- `rc4-hmac`
- `aes128-cts`
- `aes256-cts`
- `des-cbc-crc`
- `des-cbc-md5`

To configure encryption types for an Active Directory KDC:

1. Go to the Cloudera Manager Admin Console and click the **Administration** tab.
2. Select **Administration** > **Settings**.
3. Click the **Kerberos** category.
4. Locate the **Kerberos Encryption Types** and click ✚ to add the encryption types you want Active Directory to use. Make sure they are on Cloudera's list of supported `enctypes`.
5. Check the checkbox for the **Active Directory Set Encryption Types** property. This will automatically set the Cloudera Manager AD account to use the encryption types configured in the previous step.
6. Click **Save Changes** to commit the changes.

Moving Kerberos Principals to Another OU Within Active Directory

If you have a Kerberized cluster configured with an Active Directory KDC, you can use the following steps to move the Kerberos principals from one AD Organizational Unit (OU) to another.

1. Create the new OU on the Active Directory Server.
2. Use AD's Delegate Control wizard to set the permissions on the new OU such that the configured Cloudera Manager admin account has the ability to **Create, Delete and Manage User Accounts** within this OU.
3. Stop the cluster.
4. Stop the Cloudera Management Service.
5. In Active Directory, move all the Cloudera Manager and CDH components' user accounts to the new OU.
6. Go to Cloudera Manager and go to **Administration** > **Security**.
7. Go to the **Kerberos Credentials** tab and click **Configuration**.
8. Select **Scope** > **Settings**.

9. Select **Category** > **Kerberos**.
10. Locate the **Active Directory Suffix** property and edit the value to reflect the new OU name.
11. Click **Save Changes** to commit the changes.

Viewing and Regenerating Kerberos Credentials Using Cloudera Manager (MIT and AD)

Use the following instructions to regenerate the principals for your cluster.

1. Select **Administration** > **Security**.
2. The currently configured Kerberos principals are displayed under the **Kerberos Credentials** tab. If you are running HDFS, the `hdfs/hostname` and `host/hostname` principals are listed. If you are running MapReduce, the `mapred/hostname` and `host/hostname` principals are listed. The principals for other running services are also listed.
3. Only if necessary, select the principals you want to regenerate.
4. Click **Regenerate**.

Running the Security Inspector

The Security Inspector uses the Host Inspector to run a security-related set of commands on the hosts in your cluster. It reports on matters such as how Java is configured for encryption and on the default realms configured on each host:

1. Select **Administration** > **Security**.
2. Click **Security Inspector**. Cloudera Manager begins several tasks to inspect the managed hosts.
3. After the inspection completes, click **Download Result Data** or **Show Inspector Results** to review the results.

Using a Custom Kerberos Keytab Retrieval Script

The Cloudera Manager Kerberos setup procedure requires you to create an administrator account for the Cloudera Manager user. Cloudera Manager then connects to your KDC and uses this admin account to generate principals and keytabs for the remaining CDH services. If for some reason, you cannot create a Cloudera Manager administrator account on your KDC with the privileges to create other principals and keytabs for CDH services, then these will need to be created manually. Cloudera Manager gives you the option to use a custom script to retrieve keytabs from the local filesystem. To use a custom Kerberos keytab retrieval script:

1. The KDC administrators should create the required principals and keytabs, and store them securely on the Cloudera Manager Server host.
2. Create the keytab retrieval script. Your script should take two arguments: a full principal name for which it should retrieve a keytab, and a destination to which it can write the keytab. The script must be executable by the Cloudera Manager admin user, `cloudera-scm`. Depending on the principal name input by Cloudera Manager, the script should locate the corresponding keytab on the Cloudera Manager Server host (stored in step 1), and copy it into a location accessible to the `cloudera-scm` user. Here is a simple example:

```
#!/bin/bash

# Cloudera Manager will input a destination path
DEST="$1"

# Cloudera Manager will input the principal name in the format: <service>/<fqdn>@REALM
PRINC="$2"

# Assuming the '<service>_<fqdn>@REALM.keytab' naming convention for keytab files
IN=$(echo $PRINC | sed -e 's/\//_/')
SRC="/keytabs/${IN}.keytab"

# Copy the keytab to the destination input by Cloudera Manager
cp -v $SRC $DEST
```

   Note that the script will change according to the keytab naming convention followed by your organization.

3. Configure the location for the script in Cloudera Manager:

   a. Go to the Cloudera Manager Admin console.

   b. Select **Administration** > **Settings**.
   c. Select **Category** > **Kerberos**.
   d. Locate the **Custom Kerberos Keytab Retrieval Script** and set it to point to the script created in step 2.
   e. Click **Save Changes** to commit the changes.

4. Once the **Custom Kerberos Keytab Retrieval Script** property is set, whenever Cloudera Manager needs a keytab, it will ignore all other Kerberos configuration and run the keytab retrieval script to copy the required keytab to the desired destination.
5. Cloudera Manager can now distribute the keytab to all the services that need access to it.

Mapping Kerberos Principals to Short Names

Kerberos user principals typically have the format `username@REALM`, whereas Hadoop usernames are typically just `username`. To translate Kerberos principals to Hadoop usernames, Hadoop uses rules defined in the `hadoop.security.auth_to_local` property. The default setting strips the `@REALM` portion from the Kerberos principal, where `REALM` is the Kerberos realm defined by the `default_realm` setting in the NameNode `krb5.conf` file.

If you configure your cluster's Kerberos realm to trust other realms, such as a trust between your cluster's realm and a central Active Directory or MIT Kerberos realm, you must identify the trusted realms in Cloudera Manager so it can automatically generate the appropriate rules. If you do not do so, user accounts in those realms cannot access the cluster.

To specify trusted realms using Cloudera Manager:

1. Go to the **HDFS Service** > **Configuration** tab.
2. Select **Scope** > **HDFS (Service-Wide)**.
3. Select **Category** > **Security**.
4. In the Search field, type `Kerberos Realms` to find the **Trusted Kerberos Realms** and **Additional Rules to Map Kerberos Principals to Short Names** settings.
5. Add realms that are trusted by the cluster's Kerberos realm. Realm names, including Active Directory realms, must be specified in uppercase letters (for example, `CORP.EXAMPLE.COM`). To add multiple realms, use the + button.
6. Click **Save Changes**.

The auto-generated mapping rules strip the Kerberos realm (for example, `@CORP.EXAMPLE.COM`) for each realm specified in the **Trusted Kerberos Realms** setting. To customize the mapping rules, specify additional rules in the **Additional Rules to Map Kerberos Principals to Short Names** setting, one rule per line. Only enter rules in this field; Cloudera Manager automatically surrounds the rules with the appropriate XML tags for the generated `core-site.xml` file. For more information on creating custom rules, including how to translate mixed-case Kerberos principals to lowercase Hadoop usernames, see Mapping Rule Syntax on page 110.

If you specify custom mapping rules for a Kerberos realm using the **Additional Rules to Map Kerberos Principals to Short Names** setting, ensure that the same realm is not specified in the **Trusted Kerberos Realms** setting. If it is, the auto-generated rule (which only strips the realm from the principal and does no additional transformations) takes precedent, and the custom rule is ignored.

For these changes to take effect, you must restart the cluster and redeploy the client configuration. On the Cloudera Manager **Home** > **Status** tab, click the cluster-wide button and select **Deploy Client Configuration**.

Moving Kerberos Principals to Another OU Within Active Directory

If you have a Kerberized cluster configured with an Active Directory KDC, you can use the following steps to move the Kerberos principals from one AD Organizational Unit (OU) to another.

1. Create the new OU on the Active Directory Server.
2. Use AD's Delegate Control wizard to set the permissions on the new OU such that the configured Cloudera Manager admin account has the ability to **Create, Delete and Manage User Accounts** within this OU.
3. Stop the cluster.
4. Stop the Cloudera Management Service.

5. In Active Directory, move all the Cloudera Manager and CDH components' user accounts to the new OU.
6. Go to Cloudera Manager and go to **Administration** > **Security**.
7. Go to the **Kerberos Credentials** tab and click **Configuration**.
8. Select **Scope** > **Settings**.
9. Select **Category** > **Kerberos**.
10. Locate the **Active Directory Suffix** property and edit the value to reflect the new OU name.
11. Click **Save Changes** to commit the changes.

Using Auth-to-Local Rules to Isolate Cluster Users

By default, the Hadoop auth-to-local rules map a principal of the form `<username>/<hostname>@<REALM>` to `<username>`. This means if there are multiple clusters in the same realm, then principals associated with hosts of one cluster would map to the same user in all other clusters.

For example, if you have two clusters, `cluster1-host-[1..4].example.com` and `cluster2-host-[1..4].example.com`, that are part of the same Kerberos realm, `EXAMPLE.COM`, then the `cluster2` principal, `hdfs/cluster2-host1.example.com@EXAMPLE.COM`, will map to the `hdfs` user even on `cluster1` hosts.

To prevent this, use auth-to-local rules as follows to ensure only principals containing hostnames of `cluster1` are mapped to legitimate users.

1. Go to the **HDFS Service** > **Configuration** tab.
2. Select **Scope** > **HDFS (Service-Wide)**.
3. Select **Category** > **Security**.
4. In the Search field, type `Additional Rules` to find the **Additional Rules to Map Kerberos Principals to Short Names** settings.
5. Additional mapping rules can be added to the **Additional Rules to Map Kerberos Principals to Short Names** property. These rules will be inserted before the rules generated from the list of trusted realms (configured above) and before the default rule.

```
RULE:[2:$1/$2@$0](hdfs/cluster1-host1.example.com@EXAMPLE.COM)s/(.*)@EXAMPLE.COM/hdfs/
RULE:[2:$1/$2@$0](hdfs/cluster1-host2.example.com@EXAMPLE.COM)s/(.*)@EXAMPLE.COM/hdfs/
RULE:[2:$1/$2@$0](hdfs/cluster1-host3.example.com@EXAMPLE.COM)s/(.*)@EXAMPLE.COM/hdfs/
RULE:[2:$1/$2@$0](hdfs/cluster1-host4.example.com@EXAMPLE.COM)s/(.*)@EXAMPLE.COM/hdfs/
RULE:[2:$1/$2@$0](hdfs.*@EXAMPLE.COM)s/(.*)@EXAMPLE.COM/nobody/
```

   In the example, the principal `hdfs/<hostname>@REALM` is mapped to the `hdfs` user if `<hostname>` is one of the cluster hosts. Otherwise it gets mapped to `nobody`, thus ensuring that principals from other clusters do not have access to `cluster1`.

   If the cluster hosts can be represented with a regular expression, that expression can be used to make the configuration easier and more conducive to scaling. For example:

```
RULE:[2:$1/$2@$0](hdfs/cluster1-host[1-4].example.com@EXAMPLE.COM)s/(.*)@EXAMPLE.COM/hdfs/
RULE:[2:$1/$2@$0](hdfs.*@EXAMPLE.COM)s/(.*)@EXAMPLE.COM/nobody/
```

6. Click **Save Changes**.
7. Restart the HDFS service and any dependent services.

Enabling Kerberos Authentication Without the Wizard

Required Role: **Full Administrator** (for some steps), **Configurator**, or **Cluster Manager**

Whether you integrate Cloudera Manager cluster with Kerberos using the wizard or by following the manual steps below, the same requirements and pre-requisites apply. See Enabling Kerberos Authentication Using the Wizard on page 50 for details. Also see Step 4: Enabling Kerberos Using the Wizard on page 54 for Kerberos version information.

The following are the general steps for integrating Kerberos with Cloudera Manager:

Step 1: Install Cloudera Manager and CDH

Cloudera recommends setting up a fully-functional Cloudera cluster (Cloudera Manager Server, CDH, and Cloudera Manager Agent process on all cluster nodes) *before* integrating Kerberos for authentication.

User Accounts Created During Cluster Installation

During cluster installation, Cloudera Manager creates several Unix accounts and sets permissions (to various directories on the underlying host OS) that are used when integrating the cluster with Kerberos for authentication.

> **Note:** Cloudera Manager 5.3 (and later releases) can be setup for ***single user mode***. In single user mode, Hadoop users and groups are subsumed by `cloudera-scm:cloudera-scm`. Cloudera Manager starts all Cloudera Manager Agent processes and services running on the nodes in the cluster as a unit, owned by this `cloudera-scm:cloudera-scm`. Single user mode is not recommended for production clusters.

This User	Runs These Roles
`hdfs`	NameNode, DataNodes, and Secondary Node
`mapred`	JobTracker and TaskTrackers (MR1) and Job History Server (YARN)
`yarn`	ResourceManager and NodeManagers (YARN)
`oozie`	Oozie Server
`hue`	Hue Server, Beeswax Server, Authorization Manager, and Job Designer

The `hdfs` user also acts as the HDFS superuser.

When you install the Cloudera Manager Server on the server host, a new Unix user account called `cloudera-scm` is created automatically to support security. The Cloudera Manager Server uses this account to create and deploy the host principals and keytabs on your cluster.

Depending on whether you installed CDH and Cloudera Manager at the same time or not, use one of the following sections for information on configuring directory ownerships on cluster hosts:

If you installed CDH and Cloudera Manager at the Same Time

If you have a new installation and you installed CDH and Cloudera Manager at the same time, when you started the Cloudera Manager Agents on your cluster hosts, the Cloudera Manager Agent on each host automatically configured the directory owners shown in the following table to support security. Assuming the owners are configured as shown, the Hadoop daemons can then automatically set the permissions for each of the directories specified by the properties shown below to make sure they are properly restricted. It's critical that the owners are configured exactly as shown below, so do not change them:

Directory Specified in this Property	Owner
`dfs.name.dir`	`hdfs:hadoop`
`dfs.data.dir`	`hdfs:hadoop`
`mapred.local.dir`	`mapred:hadoop`
`mapred.system.dir` in HDFS	`mapred:hadoop`
`yarn.nodemanager.local-dirs`	`yarn:yarn`
`yarn.nodemanager.log-dirs`	`yarn:yarn`
`oozie.service.StoreService.jdbc.url` (if using Derby)	`oozie:oozie`
`[[database]] name`	`hue:hue`

Directory Specified in this Property	Owner
javax.jdo.option.ConnectionURL	hue:hue

If you Installed and Used CDH Before Installing Cloudera Manager

If you have been using HDFS and running MapReduce jobs in an existing installation of CDH before you installed Cloudera Manager, you must manually configure the owners of the directories shown in the table above. Doing so enables the Hadoop daemons to automatically set the permissions for each of the directories. It's critical that you manually configure the owners exactly as shown above.

Step 2: If You are Using AES-256 Encryption, Install the JCE Policy File

If you are using CentOS or RHEL 5.5 or higher, which use AES-256 encryption by default for tickets, you must install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy File on all cluster and Hadoop user hosts. There are 2 ways to do this:

- In the Cloudera Manager Admin Console, navigate to the **Hosts** page. Both, the **Add New Hosts to Cluster** wizard and the **Re-run Upgrade Wizard** will give you the option to have Cloudera Manager install the JCE Policy file for you.
- You can follow the JCE Policy File installation instructions in the `README.txt` file included in the `jce_policy-x.zip` file.

Alternatively, you can configure Kerberos to not use AES-256 by removing `aes256-cts:normal` from the `supported_enctypes` field of the `kdc.conf` or `krb5.conf` file. Note that after changing the `kdc.conf` file, you'll need to restart both the KDC and the kadmin server for those changes to take affect. You may also need to recreate or change the password of the relevant principals, including potentially the Ticket Granting Ticket principal (for example, `krbtgt/EXAMPLE.COM@EXAMPLE.COM`). If AES-256 is still used after all of those steps, it's because the `aes256-cts:normal` setting existed when the Kerberos database was created. To fix this, create a new Kerberos database and then restart both the KDC and the kadmin server.

To verify the type of encryption used in your cluster:

1. On the local KDC host, type this command in the kadmin.local or kadmin shell to create a test principal:

```
kadmin:  addprinc test
```

2. On a cluster host, type this command to start a Kerberos session as the test principal:

```
$ kinit test
```

3. After successfully running the previous command, type this command to view the encryption type in use:

```
$ klist -e
```

   If AES is being used, output like the following is displayed after you type the `klist` command (note that `AES-256` is included in the output):

```
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: test@EXAMPLE.COM
Valid starting     Expires            Service principal
05/19/11 13:25:04  05/20/11 13:25:04  krbtgt/EXAMPLE.COM@EXAMPLE.COM
    Etype (skey, tkt): AES-256 CTS mode with 96-bit SHA-1 HMAC, AES-256 CTS mode with
96-bit SHA-1 HMAC
```

Step 3: Get or Create a Kerberos Principal for the Cloudera Manager Server

In order to create and deploy the host principals and keytabs on your cluster, the Cloudera Manager Server must have the correct Kerberos principal. Specifically, the Cloudera Manager Server must have a Kerberos principal that has administrator privileges. Typically, principals with the second component of `admin` in the principal name (for example, `username/admin@EXAMPLE.COM`) have administrator privileges. This is why `admin` is shown in the following instructions and examples.

To get or create the Kerberos principal for the Cloudera Manager Server, you can do either of the following:

- Ask your Kerberos administrator to create a Kerberos administrator principal for the Cloudera Manager Server.
- Create the Kerberos principal for the Cloudera Manager Server yourself by using the following instructions in this step.

Creating the Cloudera Manager Principal

If you are using Active Directory

1. Create an Organizational Unit (OU) in your AD where all the principals used by your CDH cluster will reside.
2. Add a new AD user, for example, `<username>@EXAMPLE.COM`. The password for this user should be set to never expire.
3. Use AD's Delegate Control wizard to allow this new user to **Create, Delete and Manage User Accounts**.

If you are using MIT KDC

The instructions in this section illustrate an example of creating the Cloudera Manager Server principal for MIT Kerberos. (If you are using another version of Kerberos, refer to your Kerberos documentation for instructions.)

> **Note:** If you are running `kadmin` and the Kerberos Key Distribution Center (KDC) on the same host, use `kadmin.local` in the following steps. If the Kerberos KDC is running on a remote host, you must use `kadmin` instead of `kadmin.local`.

In the `kadmin.local` or `kadmin` shell, type the following command to create the Cloudera Manager Server principal, replacing `EXAMPLE.COM` with the name of your realm:

```
kadmin:  addprinc -pw <Password> cloudera-scm/admin@EXAMPLE.COM
```

Step 4: Import KDC Account Manager Credentials

1. In the Cloudera Manager Admin Console, select **Administration** > **Security**.
2. Go to the **Kerberos Credentials** tab and click **Import Kerberos Account Manager Credentials**.
3. In the **Import Kerberos Account Manager Credentials** dialog box, enter the username and password for the user that can create principals for CDH cluster in the KDC. This is the user/principal you created in Step 3: Get or Create a Kerberos Principal for the Cloudera Manager Server on page 69. Cloudera Manager encrypts the username and password into a keytab and uses it as needed to create new principals.

> **Note:** The username entered should have the realm portion in upper-case only as shown in the example in the UI.

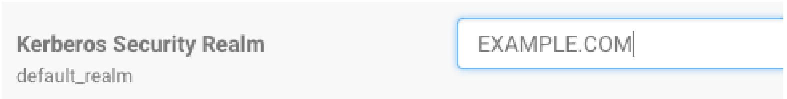
Click **Close** when complete.

Step 5: Configure the Kerberos Default Realm in the Cloudera Manager Admin Console

Minimum Required Role: Full Administrator

> **Important:** Hadoop requires a default Kerberos realm be configured in the `libdefaults` property in the `/etc/krb5.conf` file on every host in the cluster:
>
> ```
> [libdefaults]
> default_realm = EXAMPLE.COM
> ```

1. In the Cloudera Manager Admin Console, select **Administration** > **Settings**.
2. Under the Category filter, click **Kerberos**.
3. In the **Kerberos Security Realm** field, enter the name of your default Kerberos realm. Use the same realm name configured in the `/etc/krb5.conf` file. For example, `EXAMPLE.COM`:

4. Click **Save Changes**.

Step 6: Stop All Services

Minimum Required Role: **Operator** (also provided by **Configurator**, **Cluster Administrator**, **Full Administrator**)

Before you enable security in CDH, you must stop all Hadoop daemons in your cluster and then change some configuration properties. You must stop all daemons in the cluster because after one Hadoop daemon has been restarted with the configuration properties set to enable security. Daemons running without security enabled will be unable to communicate with that daemon. This requirement to stop all daemons makes it impossible to do a rolling upgrade to enable security on a Hadoop cluster.

Stop all running services, and the Cloudera Management service, as follows:

Stopping All Services

1. On the **Home** > **Status** tab, click

   to the right of the cluster name and select **Stop**.
2. Click **Stop** in the confirmation screen. The **Command Details** window shows the progress of stopping services.

   When **All services successfully stopped** appears, the task is complete and you can close the **Command Details** window.

Stopping the Cloudera Management Service

1. On the **Home** > **Status** tab, click

   to the right of **Cloudera Management Service** and select **Stop**.
2. Click **Stop** to confirm. The **Command Details** window shows the progress of stopping the roles.
3. When **Command completed with *n*/*n* successful subcommands** appears, the task is complete. Click **Close**.

Step 7: Enable Hadoop Security

Minimum Required Role: **Configurator** (also provided by **Cluster Administrator, Full Administrator**)

To enable Hadoop security for the cluster, you enable it on an HDFS service. After you do so, the Cloudera Manager Server automatically enables Hadoop security on the MapReduce and YARN services associated with that HDFS service.

1. Go to the **HDFS Service > Configuration** tab.
2. In the Search field, type **Hadoop Secure** to show the Hadoop security properties (found under the **Service-Wide** > **Security** category).
3. Click the value for the **Hadoop Secure Authentication** property and select the `kerberos` option to enable Hadoop security on the selected HDFS service.
4. Click the value for the **Hadoop Secure Authorization** property and select the checkbox to enable service-level authorization on the selected HDFS service. You can specify comma-separated lists of users and groups authorized to use Hadoop services or perform admin operations using the following properties under the **Service-Wide > Security** section:

   - **Authorized Users**: Comma-separated list of users authorized to use Hadoop services.

- **Authorized Groups**: Comma-separated list of groups authorized to use Hadoop services.
- **Authorized Admin Users**: Comma-separated list of users authorized to perform admin operations on Hadoop.
- **Authorized Admin Groups**: Comma-separated list of groups authorized to perform admin operations on Hadoop.

> **Important:** For Cloudera Manager's Monitoring services to work, the hue user should always be added as an authorized user.

5. In the Search field, type **DataNode Transceiver** to find the **DataNode Transceiver Port** property.
6. Click the value for the **DataNode Transceiver Port** property and specify a privileged port number (below 1024). Cloudera recommends 1004.

> **Note:** If there is more than one DataNode Role Group, you must specify a privileged port number for each DataNode Transceiver Port property.

7. In the Search field, type **DataNode HTTP** to find the **DataNode HTTP Web UI Port** property and specify a privileged port number (below 1024). Cloudera recommends 1006.

> **Note:** These port numbers for the two DataNode properties must be below 1024 to provide part of the security mechanism to make it impossible for a user to run a MapReduce task that impersonates a DataNode. The port numbers for the NameNode and Secondary NameNode can be anything you want, but the default port numbers are good ones to use.

8. In the Search field type **Data Directory Permissions** to find the **DataNode Data Directory Permissions** property.
9. Reset the value for the **DataNode Data Directory Permissions** property to the default value of `700` if not already set to that.
10. Make sure you have changed the **DataNode Transceiver Port**, **DataNode Data Directory Permissions** and **DataNode HTTP Web UI Port** properties for every DataNode role group.
11. Click **Save Changes** to save the configuration settings.

To enable ZooKeeper security:

1. Go to the **ZooKeeper Service > Configuration** tab and click **View and Edit**.
2. Click the value for **Enable Kerberos Authentication** property.
3. Click **Save Changes** to save the configuration settings.

To enable HBase security:

1. Go to the **HBase Service > Configuration** tab and click **View and Edit**.
2. In the Search field, type **HBase Secure** to show the Hadoop security properties (found under the **Service-Wide** > **Security** category).
3. Click the value for the **HBase Secure Authorization** property and select the checkbox to enable authorization on the selected HBase service.
4. Click the value for the **HBase Secure Authentication** property and select `kerberos` to enable authorization on the selected HBase service.
5. Click **Save Changes** to save the configuration settings.

(CDH 4.3 or later) To enable Solr security:

1. Go to the **Solr Service > Configuration** tab and click **View and Edit**.
2. In the Search field, type **Solr Secure** to show the Solr security properties (found under the **Service-Wide** > **Security** category).
3. Click the value for the **Solr Secure Authentication** property and select `kerberos` to enable authorization on the selected Solr service.
4. Click **Save Changes** to save the configuration settings.

> **Note:** If you use the Cloudera Manager Admin Console to generate a client configuration file after you enable Hadoop security on your cluster, the generated configuration file will not contain the Kerberos principal and keytab file that end users need to authenticate. Users must obtain Kerberos principal and keytab file from your Kerberos administrator and then run the `kinit` command themselves.

Step 8: Wait for the Generate Credentials Command to Finish

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

After you enable security for any of the services in Cloudera Manager, a command called Generate Credentials will be triggered automatically. You can watch the progress of the command on the top right corner of the screen that shows the running commands. Wait for this command to finish (indicated by a grey box containing "0" in it).

Step 9: Enable Hue to Work with Hadoop Security using Cloudera Manager

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

For Hue to work properly with a secure Kerberos cluster deployed with Cloudera Manager, you must add the **Kerberos Ticket Renewer** to the Hue service. You can do so in Cloudera Manager by adding a Kerberos Ticket Renewer role instance on each host with a Hue Server role.

The Hue Kerberos Ticket Renewer only renews tickets for the Hue service principal, `hue/<hostname>@<YOUR-REALM.COM>`. The Hue principal impersonates other users for applications within Hue such as the Job Browser, File Browser and so on.

Core Hadoop services such as HDFS and MapReduce do not use the Hue Kerberos Ticket Renewer. They obtain tickets at startup and use those tickets to obtain Delegation Tokens for variou access privileges. Each service handles its own ticket renewal as needed.

Adding a Kerberos Ticket Renewer role instance in Cloudera Manager:

1. Go to the **Hue** service.
2. Click the **Instances** tab.
3. Click the **Add Role Instances** button.
4. Assign the **Kerberos Ticket Renewer** role instance to the same host as the Hue server.

   When the wizard status is **Finished**, the Kerberos Ticket Renewer role instance is configured. The Hue service now works with the secure Hadoop cluster.

5. Repeat these steps for each Hue Server role.

Troubleshooting the Kerberos Ticket Renewer:

If the Hue Kerberos Ticket Renewer does not start, check the configuration of your Kerberos Key Distribution Center (KDC). Look at the ticket renewal property, `maxrenewlife`, to ensure that the principals, `hue/<hostname>` and `krbtgt`, are renewable. If these principals are not renewable, run the following commands on the KDC to enable them:

```
kadmin.local: modprinc -maxrenewlife 90day krbtgt/YOUR_REALM.COM
kadmin.local: modprinc -maxrenewlife 90day +allow_renewable hue/<hostname>@YOUR-REALM.COM
```

Step 10: (Flume Only) Use Substitution Variables for the Kerberos Principal and Keytab

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

As described in Flume security configuration, if you are using Flume on a secure cluster you must configure the HDFS sink or HBase sink with the following configuration options in the `flume.conf` file:

- `kerberosPrincipal` - fully qualified principal.
- `kerberosKeytab` - location on the local host of the keytab containing the user and host keys for the above principal

Since Cloudera Manager generates the Flume keytab files for you, and the locations of the keytab files cannot be known beforehand, substitution variables are required for Flume. Cloudera Manager provides two Flume substitution variables called $KERBEROS_PRINCIPAL and $KERBEROS_KEYTAB to configure the principal name and the keytab file path respectively on each host.

HDFS Sink Example

The following example shows an HDFS sink configuration in the `flume.conf` file (the majority of the HDFS sink configuration options have been omitted):

```
agent.sinks.sink-1.type = HDFS
agent.sinks.sink-1.hdfs.kerberosPrincipal = flume/_HOST@YOUR-REALM.COM
agent.sinks.sink-1.hdfs.kerberosKeytab = /etc/flume-ng/conf/flume.keytab
agent.sinks.sink-1.hdfs.proxyUser = weblogs
```

The text below shows the same configuration options with the substitution variables:

```
agent.sinks.sink-1.type = hdfs
agent.sinks.sink-1.hdfs.kerberosPrincipal = $KERBEROS_PRINCIPAL
agent.sinks.sink-1.hdfs.kerberosKeytab = $KERBEROS_KEYTAB
agent.sinks.sink-1.hdfs.proxyUser = weblogs
```

HBase Sink Example

The following example shows an HBase sink configuration in the `flume.conf` file (the majority of the HBase sink configuration options have been omitted):

```
agent.sinks.sink-1.type = hbase
agent.sinks.sink-1.kerberosPrincipal = flume/_HOST@YOUR-REALM.COM
agent.sinks.sink-1.kerberosKeytab = /etc/flume-ng/conf/flume.keytab
```

The text below shows the same configuration options with the substitution variables:

```
agent.sinks.sink-1.type = hbase
agent.sinks.sink-1.kerberosPrincipal = $KERBEROS_PRINCIPAL
agent.sinks.sink-1.kerberosKeytab = $KERBEROS_KEYTAB
```

Use the Flume Substitution Variables for the Kerberos Principal and Keytab

Complete the following steps to have Cloudera Manager add these variables to the `flume.conf` file on every host that Cloudera Manager manages.

1. Go to the **Flume service > Configuration** page in Cloudera Manager.
2. Click **Agent**.
3. In the **Configuration File** property, add the configuration options with the substitution variables. For example:

```
agent.sinks.sink-1.type = hdfs
agent.sinks.sink-1.hdfs.kerberosPrincipal = $KERBEROS_PRINCIPAL
agent.sinks.sink-1.hdfs.kerberosKeytab = $KERBEROS_KEYTAB
agent.sinks.sink-1.hdfs.proxyUser = weblogs
```

4. Click **Save**.

Step 11: Start All Services

Minimum Required Role: Operator (also provided by **Configurator**, **Cluster Administrator**, **Full Administrator**)

Start all services on your cluster:

Starting All Services

1. On the **Home** > **Status** tab, click

   to the right of the cluster name and select **Start**.
2. Click **Start** that appears in the next screen to confirm. The **Command Details** window shows the progress of starting services.

   When **All services successfully started** appears, the task is complete and you can close the **Command Details** window.

Starting the Cloudera Management Service

1. On the **Home** > **Status** tab, click

   to the right of **Cloudera Management Service** and select **Start**.
2. Click **Start** to confirm. The **Command Details** window shows the progress of starting the roles.
3. When **Command completed with *n/n* successful subcommands** appears, the task is complete. Click **Close**.

Step 12: Deploy Client Configurations

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

1. On the **Home** > **Status** tab, click

   to the right of the cluster name and select **Deploy Client Configuration**.
2. Click **Deploy Client Configuration**.

Step 13: Create the HDFS Superuser Principal

To be able to create home directories for users, you will need access to the HDFS superuser account. (CDH automatically created the HDFS superuser account on each cluster host during CDH installation.) When you enabled Kerberos for the HDFS service, you lost access to the default HDFS superuser account using `sudo -u hdfs` commands. Cloudera recommends you use a different user account as the superuser, not the default `hdfs` account.

Designating a Non-Default Superuser Group

To designate a different group of superusers instead of using the default `hdfs` account, follow these steps:

1. Go to the Cloudera Manager Admin Console and navigate to the HDFS service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HDFS (Service-Wide)**.
4. Select **Category** > **Security**.
5. Locate the **Superuser Group** property and change the value to the appropriate group name for your environment. For example, *<superuser>*.
6. Click **Save Changes** to commit the changes.
7. Restart the HDFS service.

   To enable your access to the superuser account now that Kerberos is enabled, you must now create a Kerberos principal or an Active Directory user whose first component is *<superuser>*:

If you are using Active Directory

Add a new user account to Active Directory, `<superuser>@YOUR-REALM.COM`. The password for this account should be set to never expire.

If you are using MIT KDC

1. In the `kadmin.local` or `kadmin` shell, type the following command to create a Kerberos principal called *<superuser>*:

```
kadmin:  addprinc <superuser>@YOUR-LOCAL-REALM.COM
```

   This command prompts you to create a password for the *<superuser>* principal. You should use a strong password because having access to this principal provides superuser access to all of the files in HDFS.

2. To run commands as the HDFS superuser, you must obtain Kerberos credentials for the *<superuser>* principal. To do so, run the following command and provide the appropriate password when prompted.

```
$ kinit <superuser>@YOUR-LOCAL-REALM.COM
```

Step 14: Get or Create a Kerberos Principal for Each User Account

Now that Kerberos is configured and enabled on your cluster, you and every other Hadoop user must have a Kerberos principal or keytab to obtain Kerberos credentials to be allowed to access the cluster and use the Hadoop services. In the next step of this procedure, you need to create your own Kerberos principals to verify that Kerberos security is working on your cluster. If you and the other Hadoop users already have a Kerberos principal or keytab, or if your Kerberos administrator can provide them, you can skip ahead to the next step.

The following instructions explain how to create a Kerberos principal for a user account.

If you are using Active Directory

Add a new AD user account, `<username>@EXAMPLE.COM` for each Cloudera Manager service that should use Kerberos authentication. The password for these service accounts should be set to never expire.

If you are using MIT KDC

1. In the kadmin.local or kadmin shell, use the following command to create a principal for your account by replacing `EXAMPLE.COM` with the name of your realm, and replacing `username` with a username:

```
kadmin:  addprinc username@EXAMPLE.COM
```

2. When prompted, enter the password twice.

Step 15: Prepare the Cluster for Each User

Before you and other users can access the cluster, there are a few tasks you must do to prepare the hosts for each user.

1. Make sure all hosts in the cluster have a Unix user account with the same name as the first component of that user's principal name. For example, the Unix account `joe` should exist on every box if the user's principal name is `joe@YOUR-REALM.COM`. You can use LDAP for this step if it is available in your organization.

   > **Note:** Each account must have a user ID that is greater than or equal to 1000. In the `/etc/hadoop/conf/taskcontroller.cfg` file, the default setting for the `banned.users` property is `mapred`, `hdfs`, and `bin` to prevent jobs from being submitted from those user accounts. The default setting for the `min.user.id` property is `1000` to prevent jobs from being submitted with a user ID less than 1000, which are conventionally Unix super users.

2. Create a subdirectory under `/user` on HDFS for each user account (for example, `/user/joe`). Change the owner and group of that directory to be the user.

```
$ hadoop fs -mkdir /user/joe
$ hadoop fs -chown joe /user/joe
```

> **Note:** `sudo -u hdfs` is not included in the commands above. This is because it is not required if Kerberos is enabled on your cluster. You will, however, need to have Kerberos credentials for the HDFS super user to successfully run these commands. For information on gaining access to the HDFS super user account, see Step 13: Create the HDFS Superuser Principal on page 75

Step 16: Verify that Kerberos Security is Working

After you have Kerberos credentials, you can verify that Kerberos security is working on your cluster by trying to run MapReduce jobs. To confirm, try launching a sleep or a pi job from the provided Hadoop examples (`/usr/lib/hadoop/hadoop-examples.jar`).

> **Note:**
>
> This section assumes you have a fully-functional CDH cluster and you have been able to access HDFS and run MapReduce jobs before you followed these instructions to configure and enable Kerberos on your cluster. If you have not already done so, you should at a minimum use the Cloudera Manager Admin Console to generate a client configuration file to enable you to access the cluster. For instructions, see Deploying Client Configuration Files.

To verify that Kerberos security is working:

1. Acquire Kerberos credentials for your user account.

```
$ kinit USERNAME@YOUR-LOCAL-REALM.COM
```

2. Enter a password when prompted.
3. Submit a sample pi calculation as a test MapReduce job. Use the following command if you use a package-based setup for Cloudera Manager:

```
$ hadoop jar /usr/lib/hadoop-0.20/hadoop-0.20.2*examples.jar pi 10 10000
Number of Maps = 10
Samples per Map = 10000
...
Job Finished in 38.572 seconds
Estimated value of Pi is 3.14120000000000000000
```

If you have a parcel-based setup, use the following command instead:

```
$ hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi
 10 10000
Number of Maps = 10
Samples per Map = 10000
...
Job Finished in 30.958 seconds
Estimated value of Pi is 3.14120000000000000000
```

You have now verified that Kerberos security is working on your cluster.

> **Important:**
>
> Running a MapReduce job will fail if you do not have a valid Kerberos ticket in your credentials cache. You can examine the Kerberos tickets currently in your credentials cache by running the `klist` command. You can obtain a ticket by running the `kinit` command and either specifying a keytab file containing credentials, or entering the password for your principal. If you do not have a valid ticket, you will receive an error such as:
>
> ```
> 11/01/04 12:08:12 WARN ipc.Client:
> Exception encountered while connecting to the server :
> javax.security.sasl.SaslException:GSS initiate failed
> [Caused by GSSException: No valid credentials provided (Mechanism level:
>  Failed to find any
> Kerberos tgt)]
> Bad connection to FS. command aborted. exception: Call to
> nn-host/10.0.0.2:8020 failed on local exception:
> java.io.IOException:javax.security.sasl.SaslException: GSS initiate
> failed
> [Caused by GSSException: No valid credentials provided
> (Mechanism level: Failed to find any Kerberos tgt)]
> ```

Step 17: (Optional) Require Authentication for HTTP Web Consoles

Required Role: Configurator, Cluster Administrator, or Full Administrator

Access to the web consoles for HTTP services, such as HDFS, MapReduce, and YARN roles can be restricted to only those users with valid Kerberos credentials. After configuring the cluster services as detailed below, the web service and the browser negotiate the authentication process using SPNEGO.

To require authentication for HTTP services on the cluster, including Web UIs:

1. From the **Clusters** tab, select the service (HDFS, MapReduce, or YARN) for which you want to enable authentication.
2. Click the **Configuration** tab.
3. Under the **Scope** filter, click ***service_name*** **(Service-Wide)**.
4. Under the **Category** filter, click **Security** to display the security configuration options.
5. In the **Enable Kerberos Authentication for HTTP Web-Consoles** setting, click the box to activate authentication requirement for the selected *service_name* (Service-Wide).
6. Click **Save Changes** to save the change.

Cloudera Manager generates new credentials for the service. When the process finishes, restart all roles for that service.

Repeat this process for the other services for which you want to require Kerberos authentication.

Configure your browser to support authentication to the HTTP services by following the appropriate steps for your browser in How to Configure Browsers for Kerberos Authentication on page 451.

Configuring Authentication for Cloudera Navigator

Cloudera Manager Server has an ***internal*** authentication mechanism, a database repository of user accounts that can be used to create user accounts. As an alternative to using the internal database, Cloudera Manager and Cloudera Navigator can be configured to use ***external*** authentication mechanisms.

Cloudera Manager Server and Cloudera Navigator each have their own user role schemes for granting privileges to system features and functions. Cloudera Manager user roles can be applied to user accounts as they are created in the internal repository. Cloudera Navigator user roles are applied to groups defined in the external system for use by Cloudera Navigator. The only user role that can be effectively applied to an account created in the Cloudera Manager internal repository is that of Navigator Administrator, which grants the user account privileges as a Full Administrator on the Cloudera Navigator services (Navigator Metadata Server, Navigator Audit Server).

In other words, assigning Cloudera Navigator user roles to user accounts requires using an external authentication mechanism.

Cloudera Navigator and External Authentication

To support its user role-based authorization scheme, Cloudera Navigator integrates with **external authentication mechanisms**. External authentication mechanisms include:

- LDAP-compliant identity/authentication services, such as Active Directory and OpenLDAP
- SAML-based SSO solutions, such as Shibboleth and SiteMinder

Cloudera Manager Server has its own internal authentication mechanism, a database repository of user accounts. However, the user accounts defined in the internal Cloudera Manager account repository cannot be assigned Cloudera Navigator user roles. The only user role that can be effectively applied to an account created in the Cloudera Manager internal repository is that of Navigator Administrator. In other words, assigning Cloudera Navigator user roles to user accounts requires using one of the external authentication mechanisms detailed in this section.

Cloudera Manager and Cloudera Navigator have their own distinct sets of user roles. Cloudera Manager and Cloudera Navigator can be configured to use external authentication mechanisms. The organization may have a central Active Directory or other LDAP-identity service used by Cloudera Manager and Cloudera Navigator for external authentication, but the relationship between each of these to the external system functions independently. That means a Cloudera Manager user that successfully authenticates to the external LDAP system cannot log in to Cloudera Manager using that same authentication token.

How it Works: Cloudera Navigator and External Authentication

At runtime, the Navigator Metadata Server role instance (the daemon) forwards login requests from Cloudera Navigator users to the external authentication mechanism which has a repository containing user accounts and groups that have been setup for Cloudera Navigator users. The groups have had specific Cloudera Navigator user role assigned to them, so once users authenticate to the external system, they can use the features of Cloudera Navigator console as specified for their group.

All this occurs transparently to Cloudera Navigator users, assuming Cloudera Navigator has been correctly configured as detailed in the appropriate section for the external mechanism—Active Directory, OpenLDAP, or SAML—as detailed in this section.

Configuring Cloudera Navigator for Active Directory

To configure Cloudera Navigator for external authentication:

1. Log in to Cloudera Manager Admin Console.
2. Select **Clusters** > **Cloudera Management Service**.
3. Click the **Configuration** tab.
4. Select **Navigator Metadata Server** for the **Scope** filter.
5. Select **External Authentication** for the **Category** filter.
6. Leave the **Authentication Backend Order** set to the default value—**Cloudera Manager Only** until *after* the external system has been successfully configured for Cloudera Navigator (as detailed in these steps) and user accounts in Active Directory instance are members of groups that have been granted Cloudera Navigator user role privileges. When Cloudera Navigator receives a login request, it checks user repositories in the order specified. Checking only the external system before having user accounts and roles configured can result in authentication failures.

   - If user accounts and groups for Cloudera Navigator already exist in the Active Directory and a group with privileges for Cloudera Manager Full Administrator or Navigator Administrator user roles contains user accounts—so that the system can be managed—the order can be set to **External then Cloudera Manager** or External Only.

7. Configure the remaining settings for the Active Directory instance as detailed in the table.

Property	Description and usage note
External Authentication Type	Active Directory

Property	Description and usage note
LDAP URL	Full path to the Active Directory instance, including the protocol specifier, `ldap` or `ldaps` (for TLS/SSL). Not necessary to specify port number if the Active Directory service is hosted using the default ports—389 (LDAP), 636 (LDAPS). For example: `ldap://ad-srv.ldap-srvs.subnet.example.com`
LDAP Bind User Distinguished Name	The user name that connects to the Active Directory service to look up login requests on behalf of Cloudera Navigator. Enter either the complete user principal name or just the short name. For example, `cn-admin@EXAMPLE.COM` or `cn-admin`. For Active Directory, this distinguished name (DN) corresponds to the `sAMAcountName`.
LDAP Bind Password	Enter the password used to log in to the Active Directory instance using the DN specified for the bind user.
Active Directory Domain	The fully-qualified domain name of the Active Directory domain controller host system. This is the service to which the bind operation For example: `ldap-servs.subnet.example.com`
LDAP Distinguished Name Pattern	Leave blank if **LDAP User Search Base** is set.
LDAP User Search Base	Specify the organizational unit (OU) and domain component (DC) properties for the LDAP search tree. For example: `ou=nav_people,dc=ldap-srvs,dc=subnet,dc=example,dc=com`
LDAP User Search Filter	Optional.
LDAP Group Search Base	`ou=nav_groups,dc=ldap-srvs,dc=subnet,dc=example,dc=com`
LDAP Group Search Filter For Logged In User	Optional.
LDAP Groups Search Filter	`(&(objectClass=groupOfNames)(cn=*{0}*))`

8. Click **Save Changes**.
9. Restart the Navigator Metadata Service:
    - From **Cloudera Management Service**, click the **Instances** tab.
    - Select **Navigator Metadata Service** from among the instances listed.
    - Click the **Actions for Selected** button and select **Restart**.

Configuring Cloudera Navigator for OpenLDAP

An LDAP-compatible identity/authentication service can be configured in a couple of different ways to find users and groups for use with Cloudera Navigator:

- Use a single base Distinguished Name (DN) (set in the **LDAP User Search Base** property) and configure the **LDAP Distinguished Name Pattern** that returns the user matching the pattern, or
- Use user and group DN patterns (entered in fields such as **LDAP User Search Filter**, **LDAP Group Search Base**, and **LDAP Group Search Filter**) to filter the LDAP lookups by specific attributes, such as organizational unit (OU).

Configuring the Server to Use OpenLDAP

1. Select **Clusters** > **Cloudera Management Service**.
2. Click the **Configuration** tab.
3. Click the **Configuration** tab.
4. Select **Scope** > **Navigator Metadata Server**.
5. Select **Category** > **External Authentication**.
6. In the **External Authentication Type**, select **LDAP**.
7. In the **LDAP URL** property, provide the URL of the LDAP server and (optionally) the base Distinguished Name (DN) (the search base) as part of the URL. For example:

```
ldap://ldap-server.corp.com/dc=corp,dc=com
```

8. In the **LDAP Bind User Distinguished Name** property, enter the LDAP account that has permission to query the LDAP database of user accounts on behalf of Cloudera Navigator.
9. In the **LDAP Bind Password** property, enter the password for the bind user entered above.
10. To use a distinguished name pattern, enter the pattern in the **LDAP Distinguished Name Pattern** property, as follows:

    - Use `{0}` to substitute the `username` field (entered by user at login) for the `uid` portion of the DN pattern. Cloudera Navigator uses the full string with the username (as uid) and passes to the external LDAP system to find and authenticate the user. For example, this pattern below searches the LDAP directory for user `foo`:

```
uid=foo,ou=people,dc=corp,dc=com
```

    - The pattern does not need to include any attributes that have been specified in LDAP User Search Base and LDAP User Search Filter properties.
    - If the **LDAP URL** property includes the base DN, only the attributes not specified in the URL path need to be the pattern only needs to be set. For example, for an LDAP URL property set to:

```
ldap://ldap-server.corp.com/dc=corp,dc=com
```

    only the `uid` and containing OU (`ou=people`) need to be specified. For example:

```
uid={0},ou=people
```

11. You can also search using User or Group search filters, using the **LDAP User Search Base**, **LDAP User Search Filter**, **LDAP Group Search Base** and **LDAP Group Search Filter** settings. These allow you to combine a base DN with a search filter to allow a greater range of search targets.

    For example, if you want to authenticate users who may be in one of multiple OUs, the search filter mechanism will allow this. You can specify the User Search Base DN as `dc=corp,dc=com` and the user search filter as `uid={0}`. Then Cloudera Navigator will search for the user anywhere in the tree starting from the Base DN. Suppose you have two OUs—`ou=Engineering` and `ou=Operations`—Cloudera Navigator will find User "foo" if it exists in either of these OUs, that is, `uid=foo,ou=Engineering,dc=corp,dc=com` or `uid=foo,ou=Operations,dc=corp,dc=com`.

    You can use a user search filter along with a DN pattern, so that the search filter provides a fallback if the DN pattern search fails.

    The Groups filters let you search to determine if a DN or username is a member of a target group. In this case, the filter you provide can be something like `member={0}` where `{0}` will be replaced with the **DN** of the user you are authenticating. For a filter requiring the username, `{1}` may be used, as `memberUid={1}`. This will return a list of groups to which the user belongs.

12. Click **Save Changes**.
13. After changing the configuration settings, restart the **Navigator Metadata Service**: click the **Instances** tab on the **Cloudera Management Service** page, check **Navigator Metadata Service**, and click **Actions for Selected** > **Restart**.

Configuring Cloudera Navigator to Use LDAPS

If the LDAP server certificate has been signed by a public CA (certificate authority), such as Symantec, following the steps below is typically not necessary.

1. Copy the CA certificate file to the Cloudera Navigator Server host.
2. Import the CA certificate from the CA certificate file to the local truststore. The default truststore is located in the `$JAVA_HOME/jre/lib/security/cacerts` file. This contains the default CA information shipped with the JDK. Create an alternate default file called `jssecacerts` in the same location as the `cacerts` file. You can now safely append CA certificates for any private or public CAs not present in the default `cacerts` file, while keeping the original file intact.

Copy the default `cacerts` file into the new `jssecacerts` file, and then importing the CA certificate to this alternate truststore.

```
$ cp $JAVA_HOME/jre/lib/security/cacerts \
  $JAVA_HOME/jre/lib/jssecacerts
```

```
$ /usr/java/latest/bin/keytool -import -alias nt_domain_name \
-keystore /usr/java/latest/jre/lib/security/jssecacerts -file path_to_cert
```

1. Configure the **LDAP URL** property to use `ldaps://ldap_server` instead of `ldap://ldap_server`.

Configuring Cloudera Navigator for SAML

Cloudera Navigator supports SAML (Security Assertion Markup Language), an XML-based open standard data format for exchanging authentication and authorization details between identity providers and service providers. One of the main benefits of SAML is that it enables Single Sign-on (SSO) for browser-based clients, such as the Cloudera Navigator console. That means that you can integrate Cloudera Navigator with SSO solutions such as Shibboleth or CA Single Sign-On (formerly, SiteMinder).

> **Note:** Not all SAML identity providers are interoperable. Cloudera Navigator has been tested with Shibboleth and SiteMinder.

Overview of SAML and SSO

The steps below assume you have a functioning SAML IDP already deployed. Here is a brief summary of some of the high level details as background to the configuration tasks:

- An ***identity provider*** or IDP is one of the main functions provided by an organization's SAML/SSO solution. The IDP provides identity assertions (tokens) to service providers that want to identify users when those users request access. service.
- A ***service provider*** or SP, such as Cloudera Navigator, protects itself from unauthorized access by checking the identity of users requesting the service against the IDP. When the SP gets back the assertion from the IDP, the service gives the requesting user the level of access for that user.
- The service's users or ***principals*** obtain access to the SP when they open their browsers to the URL of the service. Transparently to users, the SP—Cloudera Navigator, but specifically the web service hosted on the Navigator Metadata Server—sends an authentication request to the IDP through the user agent (browser) and obtains an identity assertion back from the IDP.

See the OASIS SAML Wiki for more information about SAML.

Preparing Files

You must obtain the following files and information and provide to Cloudera Navigator:

- A Java keystore containing a private key for Cloudera Navigator to use to sign/encrypt SAML messages.
- The SAML metadata XML file from your IDP. This file must contain the public certificates needed to verify the sign/encrypt key used by your IDP per the SAML Metadata Interoperability Profile.
- The entity ID that should be used to identify the Navigator Metadata Server instance.

- How the user ID is passed in the SAML authentication response:
  - As an attribute. If so, what identifier is used.
  - As the NameID.
- The method by which the Cloudera Navigator role will be established:
  - From an attribute in the authentication response:
    - What identifier will be used for the attribute
    - What values will be passed to indicate each role
  - From an external script that will be called for each use:
    - The script takes user ID as $1
    - The script must assign an exit code to reflect successful authentication of the assigned role:
      - 0 - Full Administrator
      - 1 - User Administrator
      - 2 - Auditing Viewer
      - 4 - Lineage Viewer
      - 8 - Metadata Administrator
      - 16 - Policy Viewer
      - 32 - Policy Administrator
      - 64 - Custom Metadata Administrator
      - A negative value is returned for a failure to authenticate

    To assign more than one role, add the numbers for the roles. For example, to assign the Policy Viewer and User Administrator roles, the exit code should be 17.

Configuring the Navigator Metadata Server

1. Select **Clusters** > **Cloudera Management Service**.
2. Click the **Configuration** tab.
3. Select **Navigator Metadata Server** from the **Scope** filter.
4. Select **External Authentication** from the **Category** filter.
5. Type `SAML` in the Search box to display only the SAML relevant settings.
6. Enter the values for the properties in the table based on your SAML implementation.

Property	Description and usage note
Authentication Backend Order	This property is not relevant for SAML.
External Authentication Type	SAML
Path to SAML IDP Metadata File	Set to the location (complete path) of the metadata file obtained from the IDP.
Path to SAML Keystore File	Path to the Java keystore file containing the Cloudera Navigator private key (prepared above).
SAML Keystore Password	Enter the SAML keystore password.
Alias of SAML Sign/Encrypt Private Key	Enter the alias used to identify the private key for Cloudera Navigator to use.
SAML Sign/Encrypt Private Key Password	Enter the password for the sign/encrypt private key.
SAML Entity ID	Default setting is `clouderaNavigator`. Leave set to the default unless more than one Cloudera Navigator instance is using the same IDP. Each

Property	Description and usage note
	Cloudera Navigator instance needs a unique entity ID as assigned by organizational policy.
SAML Entity Base URL	
SAML Response Binding	HTTP-Artifact (selected by default), or HTTP-Post
SAML Login URL	
Source of User ID in SAML Response	Attribute (selected by default), or NameID—Specifies the source of the user ID, an attribute or NameID. For attribute, also set the attribute name in the **SAML Attribute Identifier for User ID** property.
SAML Attribute Identifier for User ID	`urn:oid:0.9.2342.19200300.100.1.1` (Default) The standard object identifier (OID) for user IDs. This setting is used only when **Source of User ID in SAML Response** specifies Attribute.
SAML Role Assignment Mechanism	Attribute (selected by default), or Script—Specifies how user roles are assigned to authenticated user: • Attribute—Set the **SAML Attribute Identifier for User Role** and the **SAML Attribute Values for Roles** properties. • Script—A binary or shell script executable that assigns user roles. Set the path to the executable in Path to SAML Role Assignment Script.
SAML Attribute Identifier for User Role	`urn:oid:2.5.4.11` (Default) The standard OID typically used for OrganizationalUnits. Can be left to this setting.
SAML Attribute Values for Roles	Set the attribute values that will be used to indicate the user roles. For more than one role, the attribute can return comma-separated values, such as "role1, role2".
Path to SAML Role Assignment Script	Path to executable (binary or shell script) that assigns Cloudera Navigator user roles upon authentication. Required when **SAML Role Assignment Mechanism** specifies **Script**.

7. Click **Save Changes**.
8. Restart the Navigator Metadata Server role.

Configuring the Identity Provider

After Cloudera Navigator restarts, attempted logins to Cloudera Navigator are redirected to the identity provider's login page. Authentication cannot succeed until the IDP is configured for Cloudera Navigator. The configuration details are specific to the IDP, but in general you must download the SAML file from your Cloudera Navigator instance and perform the other steps below.

1. Download Cloudera Navigator's SAML metadata XML file from your Cloudera Navigator instance:

```
http://hostname:7187/saml/metadata
```

2. Inspect the metadata file and ensure that any URLs contained in the file can be resolved by users' web browsers. The IDP will redirect web browsers to these URLs at various points in the process. If the browser cannot resolve them, authentication will fail. If the URLs are incorrect, you can manually fix the XML file or set the **SAML Entity Base URL** property in the Navigator Metadata Server configuration to the correct value, and then re-download the file.
3. Provide this metadata file to your IDP using whatever mechanism your IDP provides.
4. Ensure that the IDP has access to whatever public certificates are necessary to validate the private key that was provided by Cloudera Navigator earlier.
5. Ensure that the IDP is configured to provide the User ID and Role using the attribute names that Cloudera Navigator was configured to expect, if relevant.

6. Ensure the changes to the IDP configuration have taken effect (a restart may be necessary).

Testing Cloudera Navigator and the SSO Setup

1. Return to the Cloudera Navigator home page at: `http://hostname:7187/`.
2. Attempt to log in with credentials for a user that is entitled. The authentication should complete and you should see the Home page.
3. If authentication fails, you will see an IDP provided error message. Cloudera Navigator is not involved in this part of the process, and you must ensure the IDP is working correctly to complete the authentication.
4. If authentication succeeds but the user is not authorized to use Cloudera Navigator, they will be taken to an error page that explains the situation. If a user who should be authorized sees this error, then you will need to verify their role configuration, and ensure that it is being properly communicated to the Navigator Metadata Server, whether by attribute or external script. The Cloudera Navigator log will provide details on failures to establish a user's role. If any errors occur during role mapping, Cloudera Navigator will assume the user is unauthorized.

Bypassing SAML SSO

The SAML-based SSO can be bypassed by accessing the Cloudera Navigator login page directly:

```
http://fqdn-1.example.com:7187/login.html
```

Configuring Groups for Cloudera Navigator

Cloudera Navigator user role privileges are applied to groups contained in the external authentication mechanism. In the external system using the appropriate management tool for the platform, create groups comprising all the user accounts to which a specific set of privileges should apply. For example, create one group for auditors and another group for data stewards. Create a different group for administrators. To each group, add the user accounts of people who should have the same set of Cloudera Navigator privileges. The user accounts and groups must exist in the external authentication mechanism before the user roles can be applied to them.

Once a group exists in the external authentication system, one or more Cloudera Navigator user roles can be assigned to the group using the Cloudera Navigator console. See Configuring Navigator User Roles in the Cloudera Navigator Data Management guide

Configuring Authentication in CDH Using the Command Line

The security features in CDH 5 enable Hadoop to prevent malicious user impersonation. The Hadoop daemons leverage Kerberos to perform user authentication on all remote procedure calls (RPCs). Group resolution is performed on the Hadoop master nodes, NameNode, JobTracker and ResourceManager to guarantee that group membership cannot be manipulated by users. Map tasks are run under the user account of the user who submitted the job, ensuring isolation there. In addition to these features, new authorization mechanisms have been introduced to HDFS and MapReduce to enable more control over user access to data.

The security features in CDH 5 meet the needs of most Hadoop customers because typically the cluster is accessible only to trusted personnel. In particular, Hadoop's current threat model assumes that users cannot:

1. Have `root` access to cluster machines.
2. Have `root` access to shared client machines.
3. Read or modify packets on the network of the cluster.

> **Note:**
>
> CDH 5 supports encryption of all user data sent over the network. For configuration instructions, see Configuring Encrypted Shuffle, Encrypted Web UIs, and Encrypted HDFS Transport.
>
> Note also that there is no built-in support for on-disk encryption.

Enabling Kerberos Authentication for Hadoop Using the Command Line

> **Important:**
>
> These instructions assume you know how to install and configure Kerberos, you already have a working Kerberos Key Distribution Center (KDC) and realm setup, and that you've installed the Kerberos user packages on all cluster machines and machines which will be used to access the cluster. Furthermore, Oozie and Hue require that the realm support renewable tickets. For more information about installing and configuring Kerberos, see:
>
> - MIT Kerberos Home
> - MIT Kerberos Documentation
> - Kerberos Explained
> - Microsoft Kerberos Overview
> - Microsoft Kerberos in Windows Server 2008
> - Microsoft Kerberos in Windows Server 2003

Kerberos security in CDH 5 has been tested with the following version of MIT Kerberos 5:

- krb5-1.6.1 on Red Hat Enterprise Linux 5 and CentOS 5

Kerberos security in CDH 5 is supported with the following versions of MIT Kerberos 5:

- krb5-1.6.3 on SUSE Linux Enterprise Server (SLES) 11 Service Pack 1
- krb5-1.8.1 on Ubuntu
- krb5-1.8.2 on Red Hat Enterprise Linux 6 and CentOS 6
- krb5-1.9 on Red Hat Enterprise Linux 6.1

> **Note:** The `krb5-server` package includes a `logrotate` policy file to rotate log files monthly. To take advantage of this, install the `logrotate` package. No additional configuration is necessary.

If you want to enable Kerberos SPNEGO-based authentication for the Hadoop web interfaces, see the Hadoop Auth, Java HTTP SPNEGO Documentation.

Here are the general steps to configuring secure Hadoop, each of which is described in more detail in the following sections:

Step 1: Install CDH 5

Cloudera strongly recommends that you set up a fully-functional CDH 5 cluster before you begin configuring it to use Hadoop's security features. When a secure Hadoop cluster is not configured correctly, the resulting error messages are in a preliminary state, so it's best to start implementing security after you are sure your Hadoop cluster is working properly without security.

For information about installing and configuring Hadoop and CDH 5 components, and deploying them on a cluster, see Cloudera Installation.

Step 2: Verify User Accounts and Groups in CDH 5 Due to Security

> **Note:** CDH 5 introduces a new version of MapReduce: MapReduce 2.0 (MRv2) built on the YARN framework. In this document, we refer to this new version as YARN. CDH 5 also provides an implementation of the previous version of MapReduce, referred to as MRv1 in this document.

- If you are using MRv1, see Step 2a (MRv1 only): Verify User Accounts and Groups in MRv1 on page 87 for configuration information.
- If you are using YARN, see Step 2b (YARN only): Verify User Accounts and Groups in YARN on page 88 for configuration information.

Step 2a (MRv1 only): Verify User Accounts and Groups in MRv1

> **Note:** If you are using YARN, skip this step and proceed to Step 2b (YARN only): Verify User Accounts and Groups in YARN.

During CDH 5 package installation of MRv1, the following Unix user accounts are automatically created to support security:

This User	Runs These Hadoop Programs
`hdfs`	HDFS: NameNode, DataNodes, Secondary NameNode (or Standby NameNode if you are using HA)
`mapred`	MRv1: JobTracker and TaskTrackers

The `hdfs` user also acts as the HDFS superuser.

The hadoop user no longer exists in CDH 5. If you currently use the hadoop user to run applications as an HDFS super-user, you should instead use the new `hdfs` user, or create a separate Unix account for your application such as `myhadoopapp`.

MRv1: Directory Ownership in the Local File System

Because the HDFS and MapReduce services run as different users, you must be sure to configure the correct directory ownership of the following files on the local filesystem of each host:

File System	Directory	Owner	Permissions
Local	`dfs.namenode.name.dir` (`dfs.name.dir` is deprecated but will also work)	`hdfs:hdfs`	drwx------
Local	`dfs.datanode.data.dir` (`dfs.data.dir` is deprecated but will also work)	`hdfs:hdfs`	drwx------
Local	`mapred.local.dir`	`mapred:mapred`	drwxr-xr-x

See also Deploying MapReduce v1 (MRv1) on a Cluster.

You must also configure the following permissions for the HDFS and MapReduce log directories (the default locations in `/var/log/hadoop-hdfs` and `/var/log/hadoop-0.20-mapreduce`), and the `$MAPRED_LOG_DIR/userlogs/` directory:

File System	Directory	Owner	Permissions
Local	`HDFS_LOG_DIR`	`hdfs:hdfs`	drwxrwxr-x
Local	`MAPRED_LOG_DIR`	`mapred:mapred`	drwxrwxr-x
Local	`userlogs` directory in `MAPRED_LOG_DIR`	`mapred:anygroup`	*permissions will be set automatically at daemon start time*

[1] **In CDH 5, package installation and the Hadoop daemons will automatically configure the correct permissions for you if you configure the directory ownership correctly as shown in the table above.**

MRv1: Directory Ownership on HDFS

The following directories on HDFS must also be configured as follows:

File System	Directory	Owner	Permissions
HDFS	mapreduce.jobtracker.system.dir (mapred.system.dir is deprecated but will also work)	mapred:hadoop	drwx------
HDFS	/ (root directory)	hdfs:hadoop	drwxr-xr-x

MRv1: Changing the Directory Ownership on HDFS

- If Hadoop security is enabled, use `kinit hdfs` to obtain Kerberos credentials for the `hdfs` user by running the following commands before changing the directory ownership on HDFS:

```
$ sudo -u hdfs kinit -k -t hdfs.keytab hdfs/fully.qualified.domain.name@YOUR-REALM.COM
```

If `kinit hdfs` does not work initially, run `kinit -R` after running `kinit` to obtain credentials. (For more information, see Troubleshooting Authentication Issues on page 492). To change the directory ownership on HDFS, run the following commands. Replace the example `/mapred/system` directory in the commands below with the HDFS directory specified by the `mapreduce.jobtracker.system.dir` (or `mapred.system.dir`) property in the `conf/mapred-site.xml` file:

```
$ sudo -u hdfs hadoop fs -chown mapred:hadoop /mapred/system
$ sudo -u hdfs hadoop fs -chown hdfs:hadoop /
$ sudo -u hdfs hadoop fs -chmod -R 700 /mapred/system
$ sudo -u hdfs hadoop fs -chmod 755 /
```

- In addition (whether or not Hadoop security is enabled) create the `/tmp` directory. For instructions on creating `/tmp` and setting its permissions, see these instructions.

Step 2b (YARN only): Verify User Accounts and Groups in YARN

> **Note:** If you are using MRv1, skip this step and proceed to Step 3: If you are Using AES-256 Encryption, Install the JCE Policy File on page 90.

During CDH 5 package installation of MapReduce 2.0 (YARN), the following Unix user accounts are automatically created to support security:

This User	Runs These Hadoop Programs
`hdfs`	HDFS: NameNode, DataNodes, Standby NameNode (if you are using HA)
`yarn`	YARN: ResourceManager, NodeManager
`mapred`	YARN: MapReduce JobHistory Server

> **Important:** The HDFS and YARN daemons must run as different Unix users; for example, `hdfs` and `yarn`. The MapReduce JobHistory server must run as user `mapred`. Having all of these users share a common Unix group is recommended; for example, hadoop.

[2] **When starting up, MapReduce sets the permissions for the `mapreduce.jobtracker.system.dir` (or `mapred.system.dir`) directory in HDFS, assuming the user `mapred` owns that directory.**

YARN: Directory Ownership in the Local Filesystem

Because the HDFS and MapReduce services run as different users, you must be sure to configure the correct directory ownership of the following files on the local filesystem of each host:

File System	Directory	Owner	Permissions (see [1])
Local	`dfs.namenode.name.dir` (`dfs.name.dir` is deprecated but will also work)	`hdfs:hdfs`	drwx------
Local	`dfs.datanode.data.dir` (`dfs.data.dir` is deprecated but will also work)	`hdfs:hdfs`	drwx------
Local	`yarn.nodemanager.local-dirs`	`yarn:yarn`	drwxr-xr-x
Local	`yarn.nodemanager.log-dirs`	`yarn:yarn`	drwxr-xr-x
Local	`container-executor`	`root:yarn`	--Sr-s---
Local	`conf/container-executor.cfg`	`root:yarn`	r--------

> **Important:** Configuration changes to the Linux container executor could result in local NodeManager directories (such as `usercache`) being left with incorrect permissions. To avoid this, when making changes using either Cloudera Manager or the command line, first manually remove the existing NodeManager local directories from all configured local directories (`yarn.nodemanager.local-dirs`), and let the NodeManager recreate the directory structure.

You must also configure the following permissions for the HDFS, YARN and MapReduce log directories (the default locations in `/var/log/hadoop-hdfs`, `/var/log/hadoop-yarn` and `/var/log/hadoop-mapreduce`):

File System	Directory	Owner	Permissions
Local	`HDFS_LOG_DIR`	`hdfs:hdfs`	drwxrwxr-x
Local	`$YARN_LOG_DIR`	`yarn:yarn`	drwxrwxr-x
Local	`MAPRED_LOG_DIR`	`mapred:mapred`	drwxrwxr-x

YARN: Directory Ownership on HDFS

The following directories on HDFS must also be configured as follows:

File System	Directory	Owner	Permissions
HDFS	/ (root directory)	`hdfs:hadoop`	drwxr-xr-x
HDFS	`yarn.nodemanager.remote-app-log-dir`	`yarn:hadoop`	drwxrwxrwxt
HDFS	`mapreduce.jobhistory.intermediate-done-dir`	mapred:hadoop	drwxrwxrwxt
HDFS	`mapreduce.jobhistory.done-dir`	mapred:hadoop	drwxr-x---

[3] **In CDH 5, package installation and the Hadoop daemons will automatically configure the correct permissions for you if you configure the directory ownership correctly as shown in the two tables above. See also [Deploying MapReduce v2 (YARN) on a Cluster](). **

YARN: Changing the Directory Ownership on HDFS

If Hadoop security is enabled, use `kinit hdfs` to obtain Kerberos credentials for the `hdfs` user by running the following commands:

```
$ sudo -u hdfs kinit -k -t hdfs.keytab hdfs/fully.qualified.domain.name@YOUR-REALM.COM
$ hadoop fs -chown hdfs:hadoop /
$ hadoop fs -chmod 755 /
```

If `kinit hdfs` does not work initially, run `kinit -R` after running `kinit` to obtain credentials. See Troubleshooting Authentication Issues on page 492. To change the directory ownership on HDFS, run the following commands:

```
$ sudo -u hdfs hadoop fs -chown hdfs:hadoop /
$ sudo -u hdfs hadoop fs -chmod 755 /
$ sudo -u hdfs hadoop fs -chown yarn:hadoop [yarn.nodemanager.remote-app-log-dir]
$ sudo -u hdfs hadoop fs -chmod 1777 [yarn.nodemanager.remote-app-log-dir]
$ sudo -u hdfs hadoop fs -chown mapred:hadoop [mapreduce.jobhistory.intermediate-done-dir]
$ sudo -u hdfs hadoop fs -chmod 1777 [mapreduce.jobhistory.intermediate-done-dir]
$ sudo -u hdfs hadoop fs -chown mapred:hadoop [mapreduce.jobhistory.done-dir]
$ sudo -u hdfs hadoop fs -chmod 750 [mapreduce.jobhistory.done-dir]
```

- In addition (whether or not Hadoop security is enabled) create the `/tmp` directory. For instructions on creating `/tmp` and setting its permissions, see Step 7: If Necessary, Create the HDFS /tmp Directory.
- In addition (whether or not Hadoop security is enabled), change permissions on the `/user/history` Directory. See Step 8: Create the history Directory and Set Permissions.

Step 3: If you are Using AES-256 Encryption, Install the JCE Policy File

If you are using CentOS/Red Hat Enterprise Linux 5.6 or higher, or Ubuntu, which use AES-256 encryption by default for tickets, you must install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy File on all cluster and Hadoop user machines. For JCE Policy File installation instructions, see the `README.txt` file included in the `jce_policy-x.zip` file.

Alternatively, you can configure Kerberos to not use AES-256 by removing `aes256-cts:normal` from the `supported_enctypes` field of the `kdc.conf` or `krb5.conf` file. After changing the `kdc.conf` file, you must restart both the KDC and the kadmin server for those changes to take affect. You may also need to re-create or change the password of the relevant principals, including potentially the Ticket Granting Ticket principal (krbtgt/REALM@REALM). If AES-256 is still used after completing steps, the `aes256-cts:normal` setting existed when the Kerberos database was created. To fix this, create a new Kerberos database and then restart both the KDC and the kadmin server.

To verify the type of encryption used in your cluster:

1. On the local KDC host, type this command to create a test principal:

```
$ kadmin -q "addprinc test"
```

2. On a cluster host, type this command to start a Kerberos session as test:

```
$ kinit test
```

3. On a cluster host, type this command to view the encryption type in use:

```
$ klist -e
```

If AES is being used, output like the following is displayed after you type the `klist` command; note that `AES-256` is included in the output:

```
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: test@SCM
Valid starting     Expires            Service principal
05/19/11 13:25:04  05/20/11 13:25:04  krbtgt/SCM@SCM
    Etype (skey, tkt): AES-256 CTS mode with 96-bit SHA-1 HMAC, AES-256 CTS mode with
96-bit SHA-1 HMAC
```

Step 4: Create and Deploy the Kerberos Principals and Keytab Files

A Kerberos principal is used in a Kerberos-secured system to represent a unique identity. Kerberos assigns tickets to Kerberos principals to enable them to access Kerberos-secured Hadoop services. For Hadoop, the principals should be of the format `username/fully.qualified.domain.name@YOUR-REALM.COM`. In this guide, the term `username` in the `username/fully.qualified.domain.name@YOUR-REALM.COM` principal refers to the username of an existing Unix account, such as `hdfs` or `mapred`.

A keytab is a file containing pairs of Kerberos principals and an encrypted copy of that principal's key. The keytab files are unique to each host since their keys include the hostname. This file is used to authenticate a principal on a host to Kerberos without human interaction or storing a password in a plain text file. Because having access to the keytab file for a principal allows one to act as that principal, access to the keytab files should be tightly secured. They should be readable by a minimal set of users, should be stored on local disk, and should not be included in machine backups, unless access to those backups is as secure as access to the local machine.

> **Important:**
>
> **For both MRv1 and YARN deployments:** *On every machine in your cluster*, there must be a keytab file for the `hdfs` user and a keytab file for the `mapred` user. The `hdfs` keytab file must contain entries for the `hdfs` principal and a `HTTP` principal, and the `mapred` keytab file must contain entries for the `mapred` principal and a `HTTP` principal. On each respective machine, the `HTTP` principal will be the same in both keytab files.
>
> **In addition, for YARN deployments only:** *On every machine in your cluster*, there must be a keytab file for the `yarn` user. The `yarn` keytab file must contain entries for the `yarn` principal and a `HTTP` principal. On each respective machine, the `HTTP` principal in the `yarn` keytab file will be the same as the `HTTP` principal in the `hdfs` and `mapred` keytab files.

> **Note:**
>
> The following instructions illustrate an example of creating keytab files for MIT Kerberos. If you are using another version of Kerberos, refer to your Kerberos documentation for instructions. You may use either `kadmin` or `kadmin.local` to run these commands.

When to Use kadmin.local and kadmin

When creating the Kerberos principals and keytabs, you can use kadmin.local or kadmin depending on your access and account:

- If you have root access to the KDC machine, but you do not have a Kerberos admin account, use kadmin.local.
- If you do not have root access to the KDC machine, but you do have a Kerberos admin account, use kadmin.
- If you have both root access to the KDC machine and a Kerberos admin account, you can use either one.

To start kadmin.local (on the KDC machine) or kadmin from any machine, run this command:

```
$ sudo kadmin.local
```

OR:

```
$ kadmin
```

> **Note:**
>
> In this guide, `kadmin` is shown as the prompt for commands in the kadmin shell, but you can type the same commands at the `kadmin.local` prompt in the kadmin.local shell.

> **Note:**
>
> Running kadmin.local may prompt you for a password because it is being run via sudo. You should provide your Unix password. Running kadmin may prompt you for a password because you need Kerberos admin privileges. You should provide your Kerberos admin password.

To create the Kerberos principals

> **Important:**
>
> If you plan to use Oozie, Impala, or the Hue Kerberos ticket renewer in your cluster, you must configure your KDC to allow tickets to be renewed, and you must configure `krb5.conf` to request renewable tickets. Typically, you can do this by adding the `max_renewable_life` setting to your realm in `kdc.conf`, and by adding the `renew_lifetime` parameter to the `libdefaults` section of `krb5.conf`. For more information about renewable tickets, see the Kerberos documentation.

Do the following steps for every host in your cluster. Run the commands in the kadmin.local or kadmin shell, replacing the `fully.qualified.domain.name` in the commands with the fully qualified domain name of each host. Replace `YOUR-REALM.COM` with the name of the Kerberos realm your Hadoop cluster is in.

1. In the kadmin.local or kadmin shell, create the `hdfs` principal. This principal is used for the NameNode, Secondary NameNode, and DataNodes.

```
kadmin:  addprinc -randkey hdfs/fully.qualified.domain.name@YOUR-REALM.COM
```

> **Note:**
>
> If your Kerberos administrator or company has a policy about principal names that does not allow you to use the format shown above, you can work around that issue by configuring the `<kerberos principal>` to `<short name>` mapping that is built into Hadoop. For more information, see Configuring the Mapping from Kerberos Principals to Short Names.

2. Create the `mapred` principal. If you are using MRv1, the `mapred` principal is used for the JobTracker and TaskTrackers. If you are using YARN, the `mapred` principal is used for the MapReduce Job History Server.

```
kadmin:  addprinc -randkey mapred/fully.qualified.domain.name@YOUR-REALM.COM
```

3. **YARN only:** Create the `yarn` principal. This principal is used for the ResourceManager and NodeManager.

```
kadmin:  addprinc -randkey yarn/fully.qualified.domain.name@YOUR-REALM.COM
```

4. Create the HTTP principal.

```
kadmin:  addprinc -randkey HTTP/fully.qualified.domain.name@YOUR-REALM.COM
```

> **Important:**
>
> The HTTP principal must be in the format `HTTP/fully.qualified.domain.name@YOUR-REALM.COM`. The first component of the principal must be the literal string "HTTP". This format is standard for HTTP principals in SPNEGO and is hard-coded in Hadoop. It cannot be deviated from.

To create the Kerberos keytab files

> **Important:**
>
> The instructions in this section for creating keytab files require using the Kerberos `norandkey` option in the `xst` command. If your version of Kerberos does not support the `norandkey` option, or if you cannot use `kadmin.local`, then use these alternate instructions to create appropriate Kerberos keytab files. After using those alternate instructions to create the keytab files, continue with the next section To deploy the Kerberos keytab files.

Do the following steps for every host in your cluster. Run the commands in the kadmin.local or kadmin shell, replacing the `fully.qualified.domain.name` in the commands with the fully qualified domain name of each host:

1. Create the `hdfs` keytab file that will contain the `hdfs` principal and `HTTP` principal. This keytab file is used for the NameNode, Secondary NameNode, and DataNodes.

```
kadmin:  xst -norandkey -k hdfs.keytab hdfs/fully.qualified.domain.name
HTTP/fully.qualified.domain.name
```

2. Create the `mapred` keytab file that will contain the `mapred` principal and `HTTP` principal. If you are using MRv1, the `mapred` keytab file is used for the JobTracker and TaskTrackers. If you are using YARN, the `mapred` keytab file is used for the MapReduce Job History Server.

```
kadmin:  xst -norandkey -k mapred.keytab mapred/fully.qualified.domain.name
HTTP/fully.qualified.domain.name
```

3. **YARN only:** Create the `yarn` keytab file that will contain the `yarn` principal and `HTTP` principal. This keytab file is used for the ResourceManager and NodeManager.

```
kadmin:  xst -norandkey -k yarn.keytab yarn/fully.qualified.domain.name
HTTP/fully.qualified.domain.name
```

4. Use `klist` to display the keytab file entries; a correctly-created hdfs keytab file should look something like this:

```
$ klist -e -k -t hdfs.keytab
Keytab name: WRFILE:hdfs.keytab
slot KVNO Principal
---- ---- ---------------------------------------------------------------------
   1    7    HTTP/fully.qualified.domain.name@YOUR-REALM.COM (DES cbc mode with CRC-32)

   2    7    HTTP/fully.qualified.domain.name@YOUR-REALM.COM (Triple DES cbc mode with
 HMAC/sha1)
   3    7    hdfs/fully.qualified.domain.name@YOUR-REALM.COM (DES cbc mode with CRC-32)

   4    7    hdfs/fully.qualified.domain.name@YOUR-REALM.COM (Triple DES cbc mode with
 HMAC/sha1)
```

5. Continue with the next section To deploy the Kerberos keytab files.

To deploy the Kerberos keytab files

On every node in the cluster, repeat the following steps to deploy the `hdfs.keytab` and `mapred.keytab` files. If you are using YARN, you will also deploy the `yarn.keytab` file.

1. On the host machine, copy or move the keytab files to a directory that Hadoop can access, such as `/etc/hadoop/conf`.

   a. **If you are using MRv1:**

```
$ sudo mv hdfs.keytab mapred.keytab /etc/hadoop/conf/
```

If you are using YARN:

```
$ sudo mv hdfs.keytab mapred.keytab yarn.keytab /etc/hadoop/conf/
```

b. Make sure that the `hdfs.keytab` file is only readable by the `hdfs` user, and that the `mapred.keytab` file is only readable by the `mapred` user.

```
$ sudo chown hdfs:hadoop /etc/hadoop/conf/hdfs.keytab
$ sudo chown mapred:hadoop /etc/hadoop/conf/mapred.keytab
$ sudo chmod 400 /etc/hadoop/conf/*.keytab
```

> **Note:**
>
> To enable you to use the same configuration files on every host, Cloudera recommends that you use the same name for the keytab files on every host.

c. **YARN only:** Make sure that the `yarn.keytab` file is only readable by the `yarn` user.

```
$ sudo chown yarn:hadoop /etc/hadoop/conf/yarn.keytab
$ sudo chmod 400 /etc/hadoop/conf/yarn.keytab
```

> **Important:**
>
> If the NameNode, Secondary NameNode, DataNode, JobTracker, TaskTrackers, HttpFS, or Oozie services are configured to use Kerberos HTTP SPNEGO authentication, and two or more of these services are running on the same host, then all of the running services must use the same HTTP principal and keytab file used for their HTTP endpoints.

Step 5: Shut Down the Cluster

To enable security in CDH, you must stop all Hadoop daemons in your cluster and then change some configuration properties. You must stop all daemons in the cluster because after one Hadoop daemon has been restarted with the configuration properties set to enable security, daemons running without security enabled will be unable to communicate with that daemon. This requirement to shut down all daemons makes it impossible to do a rolling upgrade to enable security on a Hadoop cluster.

To shut down the cluster, run the following command on every node in your cluster (as root):

```
$ for x in `cd /etc/init.d ; ls hadoop-*` ; do sudo service $x stop ; done
```

Step 6: Enable Hadoop Security

Cloudera recommends that all of the Hadoop configuration files throughout the cluster have the same contents.

To enable Hadoop security, add the following properties to the `core-site.xml` file *on every machine* in the cluster:

```
<property>
  <name>hadoop.security.authentication</name>
  <value>kerberos</value> <!-- A value of "simple" would disable security. -->
</property>

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

Enabling Service-Level Authorization for Hadoop Services

The `hadoop-policy.xml` file maintains access control lists (ACL) for Hadoop services. Each ACL consists of comma-separated lists of users and groups separated by a space. For example:

```
user_a,user_b group_a,group_b
```

If you only want to specify a set of users, add a comma-separated list of users followed by a blank space. Similarly, to specify only authorized groups, use a blank space at the beginning. A * can be used to give access to all users.

For example, to give users, ann, bob, and groups, `group_a`, `group_b` access to Hadoop's DataNodeProtocol service, modify the `security.datanode.protocol.acl` property in `hadoop-policy.xml`. Similarly, to give all users access to the InterTrackerProtocol service, modify `security.inter.tracker.protocol.acl` as follows:

```
<property>
    <name>security.datanode.protocol.acl</name>
    <value>ann,bob group_a,group_b</value>
    <description>ACL for DatanodeProtocol, which is used by datanodes to
    communicate with the namenode.</description>
</property>

<property>
    <name>security.inter.tracker.protocol.acl</name>
    <value>*</value>
    <description>ACL for InterTrackerProtocol, which is used by tasktrackers to
    communicate with the jobtracker.</description>
</property>
```

For more details, see Service-Level Authorization in Hadoop.

Step 7: Configure Secure HDFS

When following the instructions in this section to configure the properties in the `hdfs-site.xml` file, keep the following important guidelines in mind:

- The properties for each daemon (NameNode, Secondary NameNode, and DataNode) must specify both the HDFS and HTTP principals, as well as the path to the HDFS keytab file.
- The Kerberos principals for the NameNode, Secondary NameNode, and DataNode are configured in the `hdfs-site.xml` file. The same `hdfs-site.xml` file with *all three* of these principals must be installed on every host machine in the cluster. That is, it is not sufficient to have the NameNode principal configured on the NameNode host machine only. This is because, for example, the DataNode must know the principal name of the NameNode in order to send heartbeats to it. Kerberos authentication is bi-directional.
- The special string _HOST in the properties is replaced at run-time by the fully qualified domain name of the host machine where the daemon is running. This requires that reverse DNS is properly working on all the hosts configured this way. You may use _HOST only as the entirety of the second component of a principal name. For example, hdfs/_HOST@YOUR-REALM.COM is valid, but hdfs._HOST@YOUR-REALM.COM and hdfs/_HOST.example.com@YOUR-REALM.COM are not.
- When performing the _HOST substitution for the Kerberos principal names, the NameNode determines its own hostname based on the configured value of `fs.default.name`, whereas the DataNodes determine their hostnames based on the result of reverse DNS resolution on the DataNode hosts. Likewise, the JobTracker uses the configured value of `mapred.job.tracker` to determine its hostname whereas the TaskTrackers, like the DataNodes, use reverse DNS.
- The `dfs.datanode.address` and `dfs.datanode.http.address` port numbers for the DataNode *must* be below 1024, because this provides part of the security mechanism to make it impossible for a user to run a map task which impersonates a DataNode. The port numbers for the NameNode and Secondary NameNode can be anything you want, but the default port numbers are good ones to use.

To configure secure HDFS

Add the following properties to the `hdfs-site.xml` file *on every machine* in the cluster. Replace these example values shown below with the correct settings for your site: *path to the HDFS keytab*, *YOUR-REALM.COM*, *fully qualified domain name of NN*, and *fully qualified domain name of 2NN*

```
<!-- General HDFS security config -->
<property>
  <name>dfs.block.access.token.enable</name>
  <value>true</value>
</property>

<!-- NameNode security config -->
<property>
  <name>dfs.namenode.keytab.file</name>
  <value>/etc/hadoop/conf/hdfs.keytab</value> <!-- path to the HDFS keytab -->
</property>
<property>
  <name>dfs.namenode.kerberos.principal</name>
  <value>hdfs/_HOST@YOUR-REALM.COM</value>
</property>
<property>
  <name>dfs.namenode.kerberos.internal.spnego.principal</name>
  <value>HTTP/_HOST@YOUR-REALM.COM</value>
</property>

<!-- Secondary NameNode security config -->
<property>
  <name>dfs.secondary.namenode.keytab.file</name>
  <value>/etc/hadoop/conf/hdfs.keytab</value> <!-- path to the HDFS keytab -->
</property>
<property>
  <name>dfs.secondary.namenode.kerberos.principal</name>
  <value>hdfs/_HOST@YOUR-REALM.COM</value>
</property>
<property>
  <name>dfs.secondary.namenode.kerberos.internal.spnego.principal</name>
  <value>HTTP/_HOST@YOUR-REALM.COM</value>
</property>

<!-- DataNode security config -->
<property>
  <name>dfs.datanode.data.dir.perm</name>
  <value>700</value>
</property>
<property>
  <name>dfs.datanode.address</name>
  <value>0.0.0.0:1004</value>
</property>
<property>
  <name>dfs.datanode.http.address</name>
  <value>0.0.0.0:1006</value>
</property>
<property>
  <name>dfs.datanode.keytab.file</name>
  <value>/etc/hadoop/conf/hdfs.keytab</value> <!-- path to the HDFS keytab -->
</property>
<property>
  <name>dfs.datanode.kerberos.principal</name>
  <value>hdfs/_HOST@YOUR-REALM.COM</value>
</property>

<!-- Web Authentication config -->
<property>
  <name>dfs.web.authentication.kerberos.principal</name>
  <value>HTTP/_HOST@YOUR_REALM</value>
 </property>
```

To enable TLS/SSL for HDFS

Add the following property to `hdfs-site.xml` on *every machine* in your cluster.

```
<property>
<name>dfs.http.policy</name>
<value>HTTPS_ONLY</value>
</property>
```

Optional Step 8: Configuring Security for HDFS High Availability

CDH 5 supports the HDFS High Availability (HA) feature with Kerberos security enabled. There are two use cases that affect security for HA:

- If you are not using Quorum-based Storage (see Software Configuration for Quorum-based Storage), then no extra configuration for HA is necessary if automatic failover is not enabled. If automatic failover is enabled then access to ZooKeeper should be secured. See the Software Configuration for Shared Storage Using NFS documentation for details.
- If you are using Quorum-based Storage, then you must configure security for Quorum-based Storage by following the instructions in this section.

To configure security for Quorum-based Storage:

Add the following Quorum-based Storage configuration properties to the `hdfs-site.xml` file on all of the machines in the cluster:

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

Note:

If you already have principals and keytabs created for the machines where the JournalNodes are running, then you should reuse those principals and keytabs in the configuration properties above. You will likely have these principals and keytabs already created if you are collocating a JournalNode on a machine with another HDFS daemon.

Optional Step 9: Configure secure WebHDFS

Note:

If you are not using WebHDFS, you can skip this step.

Security for WebHDFS is disabled by default. If you want use WebHDFS with a secure cluster, this is the time to enable and configure it.

To configure secure WebHDFS:

1. If you have not already done so, enable WebHDFS by adding the following property to the `hdfs-site.xml` file *on every machine* in the cluster".

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

2. Add the following properties to the `hdfs-site.xml` file *on every machine* in the cluster. Replace the example values shown below with the correct settings for your site.

```
<property>
  <name>dfs.web.authentication.kerberos.principal</name>
  <value>HTTP/_HOST@YOUR-REALM.COM</value>
</property>

<property>
  <name>dfs.web.authentication.kerberos.keytab</name>
  <value>/etc/hadoop/conf/HTTP.keytab</value> <!-- path to the HTTP keytab -->
</property>
```

Optional Step 10: Configuring a secure HDFS NFS Gateway

To deploy a Kerberized HDFS NFS gateway, add the following configuration properties to `hdfs-site.xml` on the NFS server.

```
<property>
<name>dfs.nfs.keytab.file</name>
<value>/etc/hadoop/conf/hdfs.keytab</value> <!-- path to the HDFS or NFS gateway keytab
 -->
</property>

<property>
<name>dfs.nfs.kerberos.principal</name>
<value>hdfs/_HOST@YOUR-REALM.COM</value>
</property>
```

Potential Insecurities with a Kerberized NFS Gateway

When configuring an NFS gateway in a secure cluster, the gateway accesses the contents of HDFS using the HDFS service principals. However, authorization for end users is handled by comparing the end user's UID/GID against the UID/GID of the files on the NFS mount. No Kerberos is involved in authenticating the user first.

Because HDFS metadata doesn't have any UIDs/GIDs, only names and groups, the NFS gateway maps user names and group names to UIDs and GIDs. The user names and group names used for this mapping are derived from the local users of the host where the NFS gateway is running. The mapped IDs are then presented to the NFS client for authorization. The NFS client performs the authorization locally, comparing the UID/GID presented by the NFS Gateway to the IDs of the users on the remote host.

The main risk with this procedure is that it's quite possible to create local users with UIDs that were previously associated with any superusers. For example, users with access to HDFS can view the directories that belong to the `hdfs` user, and they can also access the underlying metadata to obtain the associated UID. Assuming the directories owned by `hdfs` have their UID set to `xyz`, a malicious user could create a new local user on the NFS gateway host with the UID set to `xyz`. This local user will now be able to freely access the `hdfs` user's files.

Solutions:

- Set the NFS Gateway property, **Allowed Hosts and Privileges**, to allow only those NFS clients that are trusted and managed by the Hadoop administrators.

  1. Go to the Cloudera Manager Admin Console and navigate to the HDFS service.
  2. Click the **Configuration** tab.
  3. Select **Scope** > **NFS Gateway**.

4. Select **Category** > **Main**.
5. Locate the **Allowed Hosts and Privileges** property and set it to a list of trusted host names and access privileges (`ro` - read-only, `rw` - read/write). For example:

```
192.168.0.0/22 rw
host1.example.org ro
```

The current default setting of this property is `* rw`, which is a security risk because it lets everybody map the NFS export in read-write mode.

6. Click **Save Changes** to commit the changes.

- Specify a user with restricted privileges for the `dfs.nfs.kerberos.principal` property, so that the NFS gateway has limited access to the NFS contents. The current default setting for this property is `hdfs/_HOST@YOUR-REALM.COM</value>`, which gives the NFS gateway unrestricted access to HDFS.

Step 11: Set Variables for Secure DataNodes

In order to allow DataNodes to start on a secure Hadoop cluster, you must set the following variables on all DataNodes in `/etc/default/hadoop-hdfs-datanode`.

```
export HADOOP_SECURE_DN_USER=hdfs
export HADOOP_SECURE_DN_PID_DIR=/var/lib/hadoop-hdfs
export HADOOP_SECURE_DN_LOG_DIR=/var/log/hadoop-hdfs
export JSVC_HOME=/usr/lib/bigtop-utils/
```

Note:

Depending on the version of Linux you are using, you may not have the `/usr/lib/bigtop-utils` directory on your system. If that is the case, set the `JSVC_HOME` variable to the `/usr/libexec/bigtop-utils` directory by using this command:

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

Step 12: Start up the NameNode

You are now ready to start the NameNode. Use the `service` command to run the `/etc/init.d` script.

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

You'll see some extra information in the logs such as:

```
10/10/25 17:01:46 INFO security.UserGroupInformation:
Login successful for user hdfs/fully.qualified.domain.name@YOUR-REALM.COM using keytab
 file /etc/hadoop/conf/hdfs.keytab
```

and:

```
12/05/23 18:18:31 INFO http.HttpServer: Adding Kerberos (SPNEGO) filter to
getDelegationToken
12/05/23 18:18:31 INFO http.HttpServer: Adding Kerberos (SPNEGO) filter to
renewDelegationToken
12/05/23 18:18:31 INFO http.HttpServer: Adding Kerberos (SPNEGO) filter to
cancelDelegationToken
12/05/23 18:18:31 INFO http.HttpServer: Adding Kerberos (SPNEGO) filter to fsck
12/05/23 18:18:31 INFO http.HttpServer: Adding Kerberos (SPNEGO) filter to getimage
12/05/23 18:18:31 INFO http.HttpServer: Jetty bound to port 50070
12/05/23 18:18:31 INFO mortbay.log: jetty-6.1.26
12/05/23 18:18:31 INFO server.KerberosAuthenticationHandler: Login using keytab
/etc/hadoop/conf/hdfs.keytab, for principal
HTTP/fully.qualified.domain.name@YOUR-REALM.COM
12/05/23 18:18:31 INFO server.KerberosAuthenticationHandler: Initialized, principal
```

```
[HTTP/fully.qualified.domain.name@YOUR-REALM.COM] from keytab
[/etc/hadoop/conf/hdfs.keytab]
```

You can verify that the NameNode is working properly by opening a web browser to `http://machine:50070/` where *machine* is the name of the machine where the NameNode is running.

Cloudera also recommends testing that the NameNode is working properly by performing a metadata-only HDFS operation, which will now require correct Kerberos credentials. For example:

```
$ hadoop fs -ls
```

Information about the kinit Command

Important:

Running the `hadoop fs -ls` command will fail if you do not have a valid Kerberos ticket in your credentials cache. You can examine the Kerberos tickets currently in your credentials cache by running the `klist` command. You can obtain a ticket by running the `kinit` command and either specifying a keytab file containing credentials, or entering the password for your principal. If you do not have a valid ticket, you will receive an error such as:

```
11/01/04 12:08:12 WARN ipc.Client: Exception encountered while connecting
 to the server : javax.security.sasl.SaslException:
 GSS initiate failed [Caused by GSSException: No valid credentials
provided (Mechanism level: Failed to find any Kerberos tgt)]
Bad connection to FS. command aborted. exception: Call to
nn-host/10.0.0.2:8020 failed on local exception: java.io.IOException:
javax.security.sasl.SaslException: GSS initiate failed [Caused by
GSSException: No valid credentials provided (Mechanism level: Failed to
 find any Kerberos tgt)]
```

Note:

The `kinit` command must either be on the path for user accounts running the Hadoop client, or else the `hadoop.kerberos.kinit.command` parameter in `core-site.xml` must be manually configured to the absolute path to the `kinit` command.

Note:

If you are running MIT Kerberos 1.8.1 or higher, a bug in versions of the Oracle JDK 6 Update 26 and higher causes Java to be unable to read the Kerberos credentials cache even after you have successfully obtained a Kerberos ticket using `kinit`. To workaround this bug, run `kinit -R` after running `kinit` initially to obtain credentials. Doing so will cause the ticket to be renewed, and the credentials cache rewritten in a format which Java can read. For more information about this problem, see Troubleshooting.

Step 12: Start up a DataNode

Begin by starting one DataNode only to make sure it can properly connect to the NameNode. Use the `service` command to run the `/etc/init.d` script.

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

You'll see some extra information in the logs such as:

```
10/10/25 17:21:41 INFO security.UserGroupInformation:
Login successful for user hdfs/fully.qualified.domain.name@YOUR-REALM.COM using keytab
 file /etc/hadoop/conf/hdfs.keytab
```

If you can get a single DataNode running and you can see it registering with the NameNode in the logs, then start up all the DataNodes. You should now be able to do all HDFS operations.

Step 14: Set the Sticky Bit on HDFS Directories

This step is optional but strongly recommended for security. In CDH 5, HDFS file permissions have support for the sticky bit. The sticky bit can be set on directories, preventing anyone except the superuser, directory owner, or file owner from deleting or moving the files within the directory. Setting the sticky bit for a file has no effect. This is useful for directories such as `/tmp` which previously had to be set to be world-writable. To set the sticky bit on the `/tmp` directory, run the following command:

```
$ sudo -u hdfs kinit -k -t hdfs.keytab hdfs/fully.qualified.domain.name@YOUR-REALM.COM
$ sudo -u hdfs hadoop fs -chmod 1777 /tmp
```

After running this command, the permissions on `/tmp` will appear as shown below. (Note the "`t`" instead of the final "`x`".)

```
$ hadoop fs -ls /
Found 2 items
drwxrwxrwt - hdfs supergroup 0 2011-02-14 15:55 /tmp
drwxr-xr-x - hdfs supergroup 0 2011-02-14 14:01 /user
```

Step 15: Start up the Secondary NameNode (if used)

At this point, you should be able to start the Secondary NameNode if you are using one:

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

> **Note:**
>
> If you are using HDFS HA, do not use the Secondary NameNode. See Configuring HDFS High Availability for instructions on configuring and deploying the Standby NameNode.

You'll see some extra information in the logs such as:

```
10/10/26 12:03:18 INFO security.UserGroupInformation:
Login successful for user hdfs/fully.qualified.domain.name@YOUR-REALM using keytab file
 /etc/hadoop/conf/hdfs.keytab
```

and:

```
12/05/23 18:33:06 INFO http.HttpServer: Adding Kerberos (SPNEGO) filter to getimage
12/05/23 18:33:06 INFO http.HttpServer: Jetty bound to port 50090
12/05/23 18:33:06 INFO mortbay.log: jetty-6.1.26
12/05/23 18:33:06 INFO server.KerberosAuthenticationHandler: Login using keytab
/etc/hadoop/conf/hdfs.keytab, for principal
HTTP/fully.qualified.domain.name@YOUR-REALM.COM
12/05/23 18:33:06 INFO server.KerberosAuthenticationHandler: Initialized, principal
[HTTP/fully.qualified.domain.name@YOUR-REALM.COM] from keytab
[/etc/hadoop/conf/hdfs.keytab]
```

You should make sure that the Secondary NameNode not only starts, but that it is successfully checkpointing.

If you're using the `service` command to start the Secondary NameNode from the `/etc/init.d scripts`, Cloudera recommends setting the property `fs.checkpoint.period` in the `hdfs-site.xml` file to a very low value (such as 5), and then monitoring the Secondary NameNode logs for a successful startup and checkpoint. Once you are satisfied that the Secondary NameNode is checkpointing properly, you should reset the `fs.checkpoint.period` to a reasonable value, or return it to the default, and then restart the Secondary NameNode.

You can make the Secondary NameNode perform a checkpoint by doing the following:

```
$ sudo -u hdfs hdfs secondarynamenode -checkpoint force
```

Note that this will not cause a running Secondary NameNode to checkpoint, but rather will start up a Secondary NameNode that will immediately perform a checkpoint and then shut down. This can be useful for debugging.

Note:

If you encounter errors during Secondary NameNode checkpointing, it may be helpful to enable Kerberos debugging output. For instructions, see Enabling Debugging Output for the Sun Kerberos Classes.

Step 16: Configure Either MRv1 Security or YARN Security

At this point, you are ready to configure either MRv1 Security or YARN Security.

- If you are using MRv1, do the steps in Configuring MRv1 Security to configure, start, and test secure MRv1.
- If you are using YARN, do the steps in Configuring YARN Security to configure, start, and test secure YARN.

Configuring MRv1 Security

If you are using YARN, skip this section and see Configuring YARN Security.

If you are using MRv1, do the following steps to configure, start, and test secure MRv1.

1. Step 1: Configure Secure MRv1 on page 102
2. Step 2: Start up the JobTracker on page 104
3. Step 3: Start up a TaskTracker on page 104
4. Step 4: Try Running a Map/Reduce Job on page 104

Step 1: Configure Secure MRv1

Keep the following important information in mind when configuring secure MapReduce:

- The properties for JobTracker and TaskTracker must specify the mapred principal, as well as the path to the `mapred` keytab file.
- The Kerberos principals for the JobTracker and TaskTracker are configured in the `mapred-site.xml` file. The same `mapred-site.xml` file with *both* of these principals must be installed on every host machine in the cluster. That is, it is not sufficient to have the JobTracker principal configured on the JobTracker host machine only. This is because, for example, the TaskTracker must know the principal name of the JobTracker to securely register with the JobTracker. Kerberos authentication is bi-directional.
- Do not use `${user.name}` in the value of the `mapred.local.dir` or `hadoop.log.dir` properties in `mapred-site.xml`. Doing so can prevent tasks from launching on a secure cluster.
- Make sure that each user who will be running MRv1 jobs exists on all cluster hosts (that is, on every host that hosts any MRv1 daemon).
- Make sure the value specified for `mapred.local.dir` is identical in `mapred-site.xml` and `taskcontroller.cfg`. If the values are different, this error message is returned.
- Make sure the value specified in `taskcontroller.cfg` for `hadoop.log.dir` is the same as what the Hadoop daemons are using, which is `/var/log/hadoop-0.20-mapreduce` by default and can be configured in `mapred-site.xml`. If the values are different, this error message is returned.

To configure secure MapReduce:

1. Add the following properties to the `mapred-site.xml` file *on every machine* in the cluster:

```
<!-- JobTracker security configs -->
<property>
  <name>mapreduce.jobtracker.kerberos.principal</name>
  <value>mapred/_HOST@YOUR-REALM.COM</value>
</property>
<property>
  <name>mapreduce.jobtracker.keytab.file</name>
  <value>/etc/hadoop/conf/mapred.keytab</value> <!-- path to the MapReduce keytab -->
</property>

<!-- TaskTracker security configs -->
<property>
  <name>mapreduce.tasktracker.kerberos.principal</name>
  <value>mapred/_HOST@YOUR-REALM.COM</value>
</property>
<property>
  <name>mapreduce.tasktracker.keytab.file</name>
  <value>/etc/hadoop/conf/mapred.keytab</value> <!-- path to the MapReduce keytab -->
</property>

<!-- TaskController settings -->
<property>
  <name>mapred.task.tracker.task-controller</name>
  <value>org.apache.hadoop.mapred.LinuxTaskController</value>
</property>
<property>
  <name>mapreduce.tasktracker.group</name>
  <value>mapred</value>
</property>
```

2. Create a file called `taskcontroller.cfg` that contains the following information:

```
hadoop.log.dir=<Path to Hadoop log directory. Should be same value used to start the
TaskTracker. This is required to set proper permissions on the log files so that they
can be written to by the user's tasks and read by the TaskTracker for serving on the
web UI.>
mapreduce.tasktracker.group=mapred
banned.users=mapred,hdfs,bin
min.user.id=1000
```

> **Note:** T
>
> he default setting for the `banned.users` property in the `taskcontroller.cfg` file is `mapred`, `hdfs`, and `bin` to prevent jobs from being submitted using those user accounts. The default setting for the `min.user.id` property is `1000` to prevent jobs from being submitted with a user ID less than 1000, which are conventionally Unix super users. Some operating systems such as CentOS 5 use a default value of 500 and above for user IDs, not 1000. If this is the case on your system, change the default setting for the `min.user.id` property to `500`. If there are user accounts on your cluster that have a user ID less than the value specified for the `min.user.id` property, the TaskTracker returns an error code of 255.

3. The path to the `taskcontroller.cfg` file is determined relative to the location of the `task-controller` binary. Specifically, the path is `<path of task-controller binary>/../../conf/taskcontroller.cfg`. If you installed the CDH 5 package, this path will always correspond to `/etc/hadoop/conf/taskcontroller.cfg`.

> **Note:**
>
> For more information about the `task-controller` program, see [Information about Other Hadoop Security Programs].

> **Important:**
>
> The same `mapred-site.xml` file and the same `hdfs-site.xml` file must both be installed on every host machine in the cluster so that the NameNode, Secondary NameNode, DataNode, JobTracker and TaskTracker can all connect securely with each other.

Step 2: Start up the JobTracker

You are now ready to start the JobTracker.

If you're using the `/etc/init.d/hadoop-0.20-mapreduce-jobtracker` script, then you can use the `service` command to run it now:

```
$ sudo service hadoop-0.20-mapreduce-jobtracker start
```

You can verify that the JobTracker is working properly by opening a web browser to `http://`*`machine`*`:50030/` where *machine* is the name of the machine where the JobTracker is running.

Step 3: Start up a TaskTracker

You are now ready to start a TaskTracker.

If you're using the `/etc/init.d/hadoop-0.20-mapreduce-tasktracker` script, then you can use the `service` command to run it now:

```
$ sudo service hadoop-0.20-mapreduce-tasktracker start
```

Step 4: Try Running a Map/Reduce Job

You should now be able to run Map/Reduce jobs. To confirm, try launching a sleep or a pi job from the provided Hadoop examples (`/usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar`). You need Kerberos credentials to do so.

> **Important:**
>
> Remember that the user who launches the job must exist on every host.

Configuring YARN Security

This page explains how to configure, start, and test secure YARN. For instructions on MapReduce1, see Configuring MRv1 Security.

1. Configure Secure YARN.
2. Start up the ResourceManager.
3. Start up the NodeManager.
4. Start up the MapReduce Job History Server.
5. Try Running a Map/Reduce YARN Job.
6. (Optional) Configure YARN for Long-running Applications

Step 1: Configure Secure YARN

Before you start:

- The Kerberos principals for the ResourceManager and NodeManager are configured in the `yarn-site.xml` file. The same `yarn-site.xml` file must be installed on every host machine in the cluster.
- Make sure that each user who runs YARN jobs exists on all cluster nodes (that is, on every node that hosts any YARN daemon).

To configure secure YARN:

1. Add the following properties to the `yarn-site.xml` file *on every machine* in the cluster:

```
<!-- ResourceManager security configs -->
<property>
  <name>yarn.resourcemanager.keytab</name>
  <value>/etc/hadoop/conf/yarn.keytab</value> <!-- path to the YARN keytab -->
</property>
<property>
  <name>yarn.resourcemanager.principal</name>
  <value>yarn/_HOST@YOUR-REALM.COM</value>
</property>

<!-- NodeManager security configs -->
<property>
  <name>yarn.nodemanager.keytab</name>
  <value>/etc/hadoop/conf/yarn.keytab</value> <!-- path to the YARN keytab -->
</property>
<property>
  <name>yarn.nodemanager.principal</name>
  <value>yarn/_HOST@YOUR-REALM.COM</value>
</property>
<property>
  <name>yarn.nodemanager.container-executor.class</name>
  <value>org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor</value>
</property>
<property>
  <name>yarn.nodemanager.linux-container-executor.group</name>
  <value>yarn</value>
</property>

<!-- To enable TLS/SSL -->
<property>
  <name>yarn.http.policy</name>
  <value>HTTPS_ONLY</value>
</property>
```

2. Add the following properties to the `mapred-site.xml` file *on every machine* in the cluster:

```
<!-- MapReduce JobHistory Server security configs -->
<property>
  <name>mapreduce.jobhistory.address</name>
  <value>host:port</value> <!-- Host and port of the MapReduce JobHistory Server; default
 port is 10020  -->
</property>
<property>
  <name>mapreduce.jobhistory.keytab</name>
  <value>/etc/hadoop/conf/mapred.keytab</value> <!-- path to the MAPRED keytab for the
 JobHistory Server -->
</property>
<property>
  <name>mapreduce.jobhistory.principal</name>
  <value>mapred/_HOST@YOUR-REALM.COM</value>
</property>

<!-- To enable TLS/SSL -->
<property>
  <name>mapreduce.jobhistory.http.policy</name>
  <value>HTTPS_ONLY</value>
</property>
```

3. Create a file called `container-executor.cfg` for the Linux Container Executor program that contains the following information:

```
yarn.nodemanager.local-dirs=<comma-separated list of paths to local NodeManager
directories. Should be same values specified in yarn-site.xml. Required to validate
paths passed to container-executor in order.>
yarn.nodemanager.linux-container-executor.group=yarn
yarn.nodemanager.log-dirs=<comma-separated list of paths to local NodeManager log
directories. Should be same values specified in yarn-site.xml. Required to set proper
permissions on the log files so that they can be written to by the user's containers
and read by the NodeManager for log aggregation.
```

```
banned.users=hdfs,yarn,mapred,bin
min.user.id=1000
```

Note:

In the `container-executor.cfg` file, the default setting for the `banned.users` property is `hdfs`, `yarn`, `mapred`, and `bin` to prevent jobs from being submitted using those user accounts. The default setting for the `min.user.id` property is `1000` to prevent jobs from being submitted with a user ID less than 1000, which are conventionally Unix super users. Some operating systems such as CentOS 5 use a default value of 500 and above for user IDs, not 1000. If this is the case on your system, change the default setting for the `min.user.id` property to `500`. If there are user accounts on your cluster that have a user ID less than the value specified for the `min.user.id` property, the NodeManager returns an error code of 255.

4. The path to the `container-executor.cfg` file is determined relative to the location of the container-executor binary. Specifically, the path is `<dirname of container-executor binary>/../etc/hadoop/container-executor.cfg`. If you installed the CDH 5 package, this path will always correspond to `/etc/hadoop/conf/container-executor.cfg`.

Note:

The `container-executor` program requires that the paths including and leading up to the directories specified in `yarn.nodemanager.local-dirs` and `yarn.nodemanager.log-dirs` to be set to 755 permissions as shown in this table on permissions on directories.

5. Verify that the ownership and permissions of the `container-executor` program corresponds to:

```
---Sr-s--- 1 root yarn 36264 May 20 15:30 container-executor
```

Note: For more information about the Linux Container Executor program, see Information about Other Hadoop Security Programs.

Step 2: Start the ResourceManager

You are now ready to start the ResourceManager.

Note: Always start ResourceManager before starting NodeManager.

If you're using the `/etc/init.d/hadoop-yarn-resourcemanager` script, then you can use the `service` command to run it now:

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

You can verify that the ResourceManager is working properly by opening a web browser to http://`host`:8088/ where `host` is the name of the machine where the ResourceManager is running.

Step 3: Start the NodeManager

You are now ready to start the NodeManager.

If you're using the `/etc/init.d/hadoop-yarn-nodemanager` script, then you can use the `service` command to run it now:

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

You can verify that the NodeManager is working properly by opening a web browser to http://`host`:8042/ where `host` is the name of the machine where the NodeManager is running.

Step 4: Start the MapReduce Job History Server

You are now ready to start the MapReduce JobHistory Server.

If you're using the `/etc/init.d/hadoop-mapreduce-historyserver` script, then you can use the `service` command to run it now:

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

You can verify that the MapReduce JobHistory Server is working properly by opening a web browser to http://`host`:19888/ where `host` is the name of the machine where the MapReduce JobHistory Server is running.

Step 5: Try Running a Map/Reduce YARN Job

You should now be able to run Map/Reduce jobs. To confirm, try launching a sleep or a pi job from the provided Hadoop examples (`/usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar`). You need Kerberos credentials to do so.

> **Important:** The user who launches the job must exist on every node.

To try running a MapReduce job using YARN, set the `HADOOP_MAPRED_HOME` environment variable and then submit the job. For example:

```
$ export HADOOP_MAPRED_HOME=/usr/lib/hadoop-mapreduce
$ /usr/bin/hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar pi 10
10000
```

Step 6: (Optional) Configure YARN for Long-running Applications

Long-running applications such as Spark Streaming jobs will need additional configuration since the default settings only allow the `hdfs` user's delegation tokens a maximum lifetime of 7 days which is not always sufficient.

You can work around this by configuring the ResourceManager as a proxy user for the corresponding HDFS NameNode so that the ResourceManager can request new tokens when the existing ones are past their maximum lifetime. YARN will then be able to continue performing localization and log-aggregation on behalf of the `hdfs` user.

Set the following property in `yarn-site.xml` to `true`:

```
<property>
<name>yarn.resourcemanager.proxy-user-privileges.enabled</name>
<value>true</value>
</property>
```

Configure the following properties in `core-site.xml` on the HDFS NameNode. You can use a more restrictive configuration by specifying hosts/groups instead of * as in the example below.

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

<property>
<name>hadoop.proxyuser.yarn.groups</name>
```

```
<value>*</value>
</property>
```

FUSE Kerberos Configuration

This section describes how to use FUSE (Filesystem in Userspace) and CDH with Kerberos security on your Hadoop cluster. FUSE enables you to mount HDFS, which makes HDFS files accessible just as if they were UNIX files.

To use FUSE and CDH with Kerberos security, follow these guidelines:

- For each HDFS user, make sure that there is a UNIX user with the same name. If there isn't, some files in the FUSE mount point will appear to be owned by a non-existent user. Although this is harmless, it can cause confusion.
- When using Kerberos authentication, users must run `kinit` before accessing the FUSE mount point. Failure to do this will result in I/O errors when the user attempts to access the mount point. For security reasons, it is not possible to list the files in the mount point without first running `kinit`.
- When a user runs `kinit`, all processes that run as that user can use the Kerberos credentials. It is not necessary to run `kinit` in the same shell as the process accessing the FUSE mount point.

Using kadmin to Create Kerberos Keytab Files

If your version of Kerberos does not support the Kerberos `-norandkey` option in the `xst` command, or if you must use `kadmin` because you cannot use `kadmin.local`, then you can use the following procedure to create Kerberos keytab files. Using the `-norandkey` option when creating keytabs is optional and a convenience, but it is not required.

Important:

For both MRv1 and YARN deployments: *On every machine in your cluster*, there must be a keytab file for the `hdfs` user and a keytab file for the `mapred` user. The `hdfs` keytab file must contain entries for the `hdfs` principal and an `HTTP` principal, and the `mapred` keytab file must contain entries for the `mapred` principal and an `HTTP` principal. On each respective machine, the `HTTP` principal will be the same in both keytab files.

In addition, for YARN deployments only: *On every machine in your cluster*, there must be a keytab file for the `yarn` user. The `yarn` keytab file must contain entries for the `yarn` principal and an `HTTP` principal. On each respective machine, the `HTTP` principal in the `yarn` keytab file will be the same as the `HTTP` principal in the `hdfs` and `mapred` keytab files.

For instructions, see To create the Kerberos keytab files on page 108.

Note:

These instructions illustrate an example of creating keytab files for MIT Kerberos. If you are using another version of Kerberos, refer to your Kerberos documentation for instructions. You can use either `kadmin` or `kadmin.local` to run these commands.

To create the Kerberos keytab files

Do the following steps for every host in your cluster, replacing the `fully.qualified.domain.name` in the commands with the fully qualified domain name of each host:

1. Create the `hdfs` keytab file, which contains an entry for the `hdfs` principal. This keytab file is used for the NameNode, Secondary NameNode, and DataNodes.

```
$ kadmin
kadmin:  xst -k hdfs-unmerged.keytab hdfs/fully.qualified.domain.name
```

2. Create the `mapred` keytab file, which contains an entry for the `mapred` principal. If you are using MRv1, the `mapred` keytab file is used for the JobTracker and TaskTrackers. If you are using YARN, the `mapred` keytab file is used for the MapReduce Job History Server.

```
kadmin:  xst -k mapred-unmerged.keytab mapred/fully.qualified.domain.name
```

3. **YARN only:** Create the `yarn` keytab file, which contains an entry for the `yarn` principal. This keytab file is used for the ResourceManager and NodeManager.

```
kadmin:  xst -k yarn-unmerged.keytab yarn/fully.qualified.domain.name
```

4. Create the `http` keytab file, which contains an entry for the HTTP principal.

```
kadmin:  xst -k http.keytab HTTP/fully.qualified.domain.name
```

5. Use the `ktutil` command to merge the previously-created keytabs:

```
$ ktutil
ktutil:  rkt hdfs-unmerged.keytab
ktutil:  rkt http.keytab
ktutil:  wkt hdfs.keytab
ktutil:  clear
ktutil:  rkt mapred-unmerged.keytab
ktutil:  rkt http.keytab
ktutil:  wkt mapred.keytab
ktutil:  clear
ktutil:  rkt yarn-unmerged.keytab
ktutil:  rkt http.keytab
ktutil:  wkt yarn.keytab
```

This procedure creates three new files: `hdfs.keytab`, `mapred.keytab` and `yarn.keytab`. These files contain entries for the `hdfs` and HTTP principals, the `mapred` and HTTP principals, and the `yarn` and HTTP principals respectively.

6. Use `klist` to display the keytab file entries. For example, a correctly-created `hdfs` keytab file should look something like this:

```
$ klist -e -k -t hdfs.keytab
Keytab name: WRFILE:hdfs.keytab
slot KVNO Principal
---- ---- ---------------------------------------------------------------------
   1    7    HTTP/fully.qualified.domain.name@YOUR-REALM.COM (DES cbc mode with CRC-32)

   2    7    HTTP/fully.qualified.domain.name@YOUR-REALM.COM (Triple DES cbc mode with
 HMAC/sha1)
   3    7    hdfs/fully.qualified.domain.name@YOUR-REALM.COM (DES cbc mode with CRC-32)

   4    7    hdfs/fully.qualified.domain.name@YOUR-REALM.COM (Triple DES cbc mode with
 HMAC/sha1)
```

7. To verify that you have performed the merge procedure correctly, make sure you can obtain credentials as both the `hdfs` and HTTP principals using the single merged keytab:

```
$ kinit -k -t hdfs.keytab hdfs/fully.qualified.domain.name@YOUR-REALM.COM
$ kinit -k -t hdfs.keytab HTTP/fully.qualified.domain.name@YOUR-REALM.COM
```

If either of these commands fails with an error message such as "`kinit: Key table entry not found while getting initial credentials`", then something has gone wrong during the merge procedure. Go back to step 1 of this document and verify that you performed all the steps correctly.

8. To continue the procedure of configuring Hadoop security in CDH 5, follow the instructions in the section To deploy the Kerberos keytab files.

Configuring the Mapping from Kerberos Principals to Short Names

You configure the mapping from Kerberos principals to short names in the `hadoop.security.auth_to_local` property setting in the `core-site.xml` file. Kerberos has this support natively, and Hadoop's implementation reuses Kerberos's configuration language to specify the mapping.

A mapping consists of a set of rules that are evaluated in the order listed in the `hadoop.security.auth_to_local` property. The first rule that matches a principal name is used to map that principal name to a short name. Any later rules in the list that match the same principal name are ignored.

You specify the mapping rules on separate lines in the `hadoop.security.auth_to_local` property as follows:

```
<property>
  <name>hadoop.security.auth_to_local</name>
  <value>
  RULE:[<principal translation>](<acceptance filter>)<short name substitution>
  RULE:[<principal translation>](<acceptance filter>)<short name substitution>
  DEFAULT
  </value>
</property>
```

Mapping Rule Syntax

To specify a mapping rule, use the prefix string `RULE:` followed by three sections—principal translation, acceptance filter, and short name substitution—described in more detail below. The syntax of a mapping rule is:

```
RULE:[<principal translation>](<acceptance filter>)<short name substitution>
```

Principal Translation

The first section of a rule, `<principal translation>`, performs the matching of the principal name to the rule. If there is a match, the principal translation also does the initial translation of the principal name to a short name. In the `<principal translation>` section, you specify the number of components in the principal name and the pattern you want to use to translate those principal component(s) and realm into a short name. In Kerberos terminology, a principal name is a set of components separated by slash ("/") characters.

The principal translation is composed of two parts that are both specified within "[ ]" using the following syntax:

```
[<number of components in principal name>:<initial specification of short name>]
```

where:

<number of components in principal name> – This first part specifies the number of components in the principal name (not including the realm) and must be 1 or 2. A value of 1 specifies principal names that have a single component (for example, `hdfs`), and 2 specifies principal names that have two components (for example, `hdfs/fully.qualified.domain.name`). A principal name that has only one component will only match single-component rules, and a principal name that has two components will only match two-component rules.

<initial specification of short name> – This second part specifies a pattern for translating the principal component(s) and the realm into a short name. The variable $0 translates the realm, $1 translates the first component, and $2 translates the second component.

Here are some examples of principal translation sections. These examples use `atm@YOUR-REALM.COM` and `atm/fully.qualified.domain.name@YOUR-REALM.COM` as principal name inputs:

This Principal Translation	Translates atm@YOUR-REALM.COM into this short name	Translates atm/fully.qualified.domain.name@YOUR-REALM.COM into this short name
`[1:$1@$0]`	atm@YOUR-REALM.COM	**Rule does not match**[1]
`[1:$1]`	atm	**Rule does not match**[1]
`[1:$1.foo]`	atm.foo	**Rule does not match**[1]

This Principal Translation	Translates atm@YOUR-REALM.COM into this short name	Translates atm/fully.qualified.domain.name@YOUR-REALM.COM into this short name
`[2:$1/$2@$0]`	**Rule does not match**[2]	atm/fully.qualified.domain.name@YOUR-REALM.COM
`[2:$1/$2]`	**Rule does not match**[2]	atm/fully.qualified.domain.name
`[2:$1@$0]`	**Rule does not match**[2]	atm@YOUR-REALM.COM
`[2:$1]`	**Rule does not match**[2]	atm

Footnotes:

[1]Rule does not match because there are two components in principal name `atm/fully.qualified.domain.name@YOUR-REALM.COM`

[2]Rule does not match because there is one component in principal name `atm@YOUR-REALM.COM`

Acceptance Filter

The second section of a rule, (`<acceptance filter>`), matches the translated short name from the principal translation (that is, the output from the first section). The acceptance filter is specified in "( )" characters and is a standard regular expression. A rule matches only if the specified regular expression matches the entire translated short name from the principal translation. That is, there's an implied ^ at the beginning of the pattern and an implied $ at the end.

Short Name Substitution

The third and final section of a rule is the (`<short name substitution>`). If there is a match in the second section, the acceptance filter, the (`<short name substitution>`) section does a final translation of the short name from the first section. This translation is a sed replacement expression (`s/.../.../g`) that translates the short name from the first section into the final short name string. The short name substitution section is optional. In many cases, it is sufficient to use the first two sections only.

Converting Principal Names to Lowercase

In some organizations, naming conventions result in mixed-case usernames (for example, `John.Doe`) or even uppercase usernames (for example, `JDOE`) in Active Directory or LDAP. This can cause a conflict when the Linux username and HDFS home directory are lowercase.

To convert principal names to lowercase, append `/L` to the rule.

Example Rules

Suppose all of your service principals are either of the form `App.service-name/fully.qualified.domain.name@YOUR-REALM.COM` or `App.service-name@YOUR-REALM.COM`, and you want to map these to the short name string `service-name`. To do this, your rule set would be:

```
<property>
  <name>hadoop.security.auth_to_local</name>
  <value>
 RULE:[1:$1](App\..*)s/App\.(.*)/$1/g
  RULE:[2:$1](App\..*)s/App\.(.*)/$1/g
  DEFAULT
  </value>
</property>
```

The first $1 in each rule is a reference to the first component of the full principal name, and the second $1 is a regular expression back-reference to text that is matched by `(.*)`.

In the following example, suppose your company's naming scheme for user accounts in Active Directory is FirstnameLastname (for example, JohnDoe), but user home directories in HDFS are /user/firstnamelastname. The following rule set converts user accounts in the CORP.EXAMPLE.COM domain to lowercase.

```
<property>
  <name>hadoop.security.auth_to_local</name>
  <value>RULE:[1:$1@$0](.*@\QCORP.EXAMPLE.COM\E$)s/@\QCORP.EXAMPLE.COM\E$///L
RULE:[2:$1@$0](.*@\QCORP.EXAMPLE.COM\E$)s/@\QCORP.EXAMPLE.COM\E$///L
DEFAULT</value>
</property>
```

In this example, the JohnDoe@CORP.EXAMPLE.COM principal becomes the johndoe HDFS user.

Default Rule

You can specify an optional default rule called DEFAULT (see example above). The default rule reduces a principal name down to its first component only. For example, the default rule reduces the principal names atm@YOUR-REALM.COM or atm/fully.qualified.domain.name@YOUR-REALM.COM down to atm, assuming that the default domain is YOUR-REALM.COM.

The default rule applies only if the principal is in the default realm.

If a principal name does not match any of the specified rules, the mapping for that principal name will fail.

Testing Mapping Rules

You can test mapping rules for a long principal name by running:

```
$ hadoop org.apache.hadoop.security.HadoopKerberosName name1 name2 name3
```

Enabling Debugging Output for the Sun Kerberos Classes

Initially getting a secure Hadoop cluster configured properly can be tricky, especially for those who are not yet familiar with Kerberos. To help with this, it can be useful to enable debugging output for the Sun Kerberos classes. To do so, set the HADOOP_OPTS environment variable to the following:

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

Configuring Authentication for Other Components

The pages in this section provide configuration steps for many components that may or may not be configured by default during Kerberos configuration. Some components use native authentication rather than or in addition to Kerberos.

Flume Authentication

Flume agents have the ability to store data on an HDFS filesystem configured with Hadoop security. The Kerberos system and protocols authenticate communications between clients and services. Hadoop clients include users and MapReduce jobs on behalf of users, and the services include HDFS and MapReduce. Flume acts as a Kerberos principal (user) and needs Kerberos credentials to interact with the Kerberos security-enabled service. Authenticating a user or a service can be done using a Kerberos keytab file. This file contains a key that is used to obtain a ticket-granting ticket (TGT). The TGT is used to mutually authenticate the client and the service using the Kerberos KDC.

The following sections describe how to use Flume 1.3.x and CDH 5 with Kerberos security on your Hadoop cluster:

> **Important:**
>
> To enable Flume to work with Kerberos security on your Hadoop cluster, make sure you perform the installation and configuration steps in Configuring Hadoop Security in CDH 5.

> **Note:**
>
> These instructions have been tested with CDH 5 and MIT Kerberos 5 only. The following instructions describe an example of how to configure a Flume agent to be a client as the user `flume` to a secure HDFS service. This section does not describe how to secure the communications between Flume agents, which is not currently implemented.

Configuring Flume's Security Properties

Contents:

Writing as a single user for all HDFS sinks in a given Flume agent

The Hadoop services require a three-part principal that has the form of `username/fully.qualified.domain.name@YOUR-REALM.COM`. Cloudera recommends using `flume` as the first component and the fully qualified domain name of the host machine as the second. Assuming that Kerberos and security-enabled Hadoop have been properly configured on the Hadoop cluster itself, you must add the following parameters to the Flume agent's `flume.conf` configuration file, which is typically located at `/etc/flume-ng/conf/flume.conf`:

```
agentName.sinks.sinkName.hdfs.kerberosPrincipal =
flume/fully.qualified.domain.name@YOUR-REALM.COM
agentName.sinks.sinkName.hdfs.kerberosKeytab = /etc/flume-ng/conf/flume.keytab
```

where:

`agentName` is the name of the Flume agent being configured, which in this release defaults to the value "agent". `sinkName` is the name of the HDFS sink that is being configured. The respective sink's `type` must be `HDFS`. These properties can also be set using the substitution strings *$KERBEROS_PRINCIPAL* and *$KERBEROS_KEYTAB*, respectively.

In the previous example, `flume` is the first component of the principal name, `fully.qualified.domain.name` is the second, and `YOUR-REALM.COM` is the name of the Kerberos realm your Hadoop cluster is in. The `/etc/flume-ng/conf/flume.keytab` file contains the keys necessary for `flume/fully.qualified.domain.name@YOUR-REALM.COM` to authenticate with other services.

Flume and Hadoop also provide a simple keyword, `_HOST`, that gets expanded to be the fully qualified domain name of the host machine where the service is running. This allows you to have one `flume.conf` file with the same `hdfs.kerberosPrincipal` value on all of your agent host machines.

```
agentName.sinks.sinkName.hdfs.kerberosPrincipal = flume/_HOST@YOUR-REALM.COM
```

Writing as different users across multiple HDFS sinks in a single Flume agent

In this release, support has been added for secure impersonation of Hadoop users (similar to "sudo" in UNIX). This is implemented in a way similar to how Oozie implements secure user impersonation.

The following steps to set up secure impersonation from Flume to HDFS assume your cluster is configured using Kerberos. (However, impersonation also works on non-Kerberos secured clusters, and Kerberos-specific aspects should be omitted in that case.)

1. Configure Hadoop to allow impersonation. Add the following configuration properties to your `core-site.xml`.

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

```
 a user</description>
</property>
```

You can use the wildcard character * to enable impersonation of any user from any host. For more information, see Secure Impersonation.

2. Set up a Kerberos keytab for the Kerberos principal and host Flume is connecting to HDFS from. This user must match the Hadoop configuration in the preceding step. For instructions, see Configuring Hadoop Security in CDH 5.
3. Configure the HDFS sink with the following configuration options:
4. `hdfs.kerberosPrincipal` - fully qualified principal. Note: `_HOST` will be replaced by the hostname of the local machine (only in-between the / and @ characters)
5. `hdfs.kerberosKeytab` - location on the local machine of the keytab containing the user and host keys for the above principal
6. `hdfs.proxyUser` - the proxy user to impersonate

Example snippet (the majority of the HDFS sink configuration options have been omitted):

```
agent.sinks.sink-1.type = HDFS
agent.sinks.sink-1.hdfs.kerberosPrincipal = flume/_HOST@YOUR-REALM.COM
agent.sinks.sink-1.hdfs.kerberosKeytab = /etc/flume-ng/conf/flume.keytab
agent.sinks.sink-1.hdfs.proxyUser = weblogs

agent.sinks.sink-2.type = HDFS
agent.sinks.sink-2.hdfs.kerberosPrincipal = flume/_HOST@YOUR-REALM.COM
agent.sinks.sink-2.hdfs.kerberosKeytab = /etc/flume-ng/conf/flume.keytab
agent.sinks.sink-2.hdfs.proxyUser = applogs
```

In the above example, the flume Kerberos principal impersonates the user `weblogs` in `sink-1` and the user `applogs` in `sink-2`. This will only be allowed if the Kerberos KDC authenticates the specified principal (`flume` in this case), and the if NameNode authorizes impersonation of the specified proxy user by the specified principal.

Limitations

At this time, Flume does not support using multiple Kerberos principals or keytabs in the same agent. Therefore, if you want to create files as multiple users on HDFS, then impersonation must be configured, and exactly one principal must be configured in Hadoop to allow impersonation of all desired accounts. In addition, the same keytab path must be used across all HDFS sinks in the same agent. If you attempt to configure multiple principals or keytabs in the same agent, Flume will emit the following error message:

```
Cannot use multiple kerberos principals in the same agent. Must restart agent to use
new principal or keytab.
```

Configuring Kerberos for Flume Thrift Source and Sink Using Cloudera Manager

The Thrift source can be configured to start in secure mode by enabling Kerberos authentication. To communicate with a secure Thrift source, the Thrift sink should also be operating in secure mode.

1. Open the Cloudera Manager Admin Console and go to the **Flume** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Agent**.
4. Select **Category** > **Main**.
5. Edit the **Configuration File** property and add the Thrift source and sink properties listed in the tables below to the configuration file.

Table 1: Thrift Source Properties

Property	Description
`kerberos`	Set to `true` to enable Kerberos authentication. The `agent-principal` and `agent-keytab` properties are required

Property	Description
	for successful authentication. The Thrift source in secure mode, will accept connections only from Thrift sinks that have Kerberos-enabled and are successfully authenticated to the KDC.
`agent-principal`	The Kerberos principal used by the Thrift Source to authenticate to the KDC.
`agent-keytab`	The path to the keytab file used by the Thrift Source in combination with the `agent-principal` to authenticate to the KDC.

Table 2: Thrift Sink Properties

Property	Description
`kerberos`	Set to `true` to enable Kerberos authentication. In Kerberos mode, client-principal, client-keytab and server-principal are required for successful authentication and communication to a Kerberos enabled Thrift Source.
`client-principal`	The principal used by the Thrift Sink to authenticate to the Kerberos KDC.
`client-keytab`	The path to the keytab file used by the Thrift Sink in combination with the `client-principal` to authenticate to the KDC.
`server-principal`	The principal of the Thrift Source to which this Thrift Sink connects.

> **Note:** Since Cloudera Manager generates the Flume keytab files for you, and the locations of the keytab files cannot be known beforehand, substitution variables are required for Flume. Cloudera Manager provides two Flume substitution variables called `$KERBEROS_PRINCIPAL` and $KERBEROS_KEYTAB to configure the principal name and the keytab file path respectively on each host.

Make sure you are configuring these properties for ***each*** Thrift source and sink instance managed by Cloudera Manager. For example, for agent `a1`, source `r1`, and sink `k1`, you would add the following properties:

```
# Kerberos properties for Thrift source s1
a1.sources.r1.kerberos=true
a1.sources.r1.agent-principal=<source_principal>
a1.sources.r1.agent-keytab=<path/to/source/keytab>

# Kerberos properties for Thrift sink k1
a1.sinks.k1.kerberos=true
a1.sinks.k1.client-principal=<sink_principal>
a1.sinks.k1.client-keytab=<path/to/sink/keytab>
a1.sinks.k1.server-principal=<path/to/source/keytab>
```

6. Click **Save Changes** to commit the changes.
7. Restart the Flume service.

Configuring Kerberos for Flume Thrift Source and Sink Using the Command Line

The Thrift source can be configured to start in secure mode by enabling Kerberos authentication. To communicate with a secure Thrift source, the Thrift sink should also be operating in secure mode.

The following tables list the properties that must be configured in the `/etc/flume-ng/conf/flume.conf` file to enable Kerberos for Flume's Thrift source and sink instances.

Table 3: Thrift Source Properties

Property	Description
`kerberos`	Set to `true` to enable Kerberos authentication. The `agent-principal` and `agent-keytab` properties are required for successful authentication. The Thrift source in secure mode, will accept connections only from Thrift sinks that have Kerberos-enabled and are successfully authenticated to the KDC.
`agent-principal`	The Kerberos principal used by the Thrift Source to authenticate to the KDC.
`agent-keytab`	The path to the keytab file used by the Thrift Source in combination with the `agent-principal` to authenticate to the KDC.

Table 4: Thrift Sink Properties

Property	Description
`kerberos`	Set to `true` to enable Kerberos authentication. In Kerberos mode, client-principal, client-keytab and server-principal are required for successful authentication and communication to a Kerberos enabled Thrift Source.
`client-principal`	The principal used by the Thrift Sink to authenticate to the Kerberos KDC.
`client-keytab`	The path to the keytab file used by the Thrift Sink in combination with the `client-principal` to authenticate to the KDC.
`server-principal`	The principal of the Thrift Source to which this Thrift Sink connects.

Make sure you are configuring these properties for ***each*** Thrift source and sink instance. For example, for agent `a1`, source `r1`, and sink `k1`, you would add the following properties:

```
# Kerberos properties for Thrift source s1
a1.sources.r1.kerberos=true
a1.sources.r1.agent-principal=<source_principal>
a1.sources.r1.agent-keytab=<path/to/source/keytab>

# Kerberos properties for Thrift sink k1
a1.sinks.k1.kerberos=true
a1.sinks.k1.client-principal=<sink_principal>
a1.sinks.k1.client-keytab=<path/to/sink/keytab>
a1.sinks.k1.server-principal=<path/to/source/keytab>
```

Configure these sets of properties for as many instances of the Thrift source and sink as needed to enable Kerberos.

Flume Account Requirements

This section provides an overview of the account and credential requirements for Flume to write to a Kerberized HDFS. Note the distinctions between the Flume agent machine, DataNode machine, and NameNode machine, as well as the `flume` Unix user account versus the `flume` Hadoop/Kerberos user account.

- Each Flume agent machine that writes to HDFS (using a configured HDFS sink) needs a Kerberos principal of the form:

```
flume/fully.qualified.domain.name@YOUR-REALM.COM
```

  where `fully.qualified.domain.name` is the fully qualified domain name of the given Flume agent host machine, and `YOUR-REALM.COM` is the Kerberos realm.

- Each Flume agent machine that writes to HDFS does *not* need to have a `flume` Unix user account to write files owned by the `flume` Hadoop/Kerberos user. Only the keytab for the `flume` Hadoop/Kerberos user is required on the Flume agent machine.
- DataNode machines do *not* need Flume Kerberos keytabs and also do *not* need the `flume` Unix user account.
- TaskTracker (MRv1) or NodeManager (YARN) machines need a `flume` Unix user account *if and only if* MapReduce jobs are being run as the `flume` Hadoop/Kerberos user.
- The NameNode machine needs to be able to resolve the groups of the `flume` user. The groups of the `flume` user on the NameNode machine are mapped to the Hadoop groups used for authorizing access.
- The NameNode machine does *not* need a Flume Kerberos keytab.

Testing the Flume HDFS Sink Configuration

To test whether your Flume HDFS sink is properly configured to connect to your secure HDFS cluster, you must run data through Flume. An easy way to do this is to configure a Netcat source, a Memory channel, and an HDFS sink. Start Flume with that configuration, and use the nc command (available freely online and with many UNIX distributions) to send events to the Netcat source port. The resulting events should appear on HDFS in the configured location. If the events do not appear, check the Flume log at `/var/log/flume-ng/flume.log` for any error messages related to Kerberos.

Writing to a Secure HBase Cluster

Before you write to a secure HBase cluster, be aware of the following:

- Flume must be configured to use Kerberos security as documented above, and HBase must be configured to use Kerberos security as documented in [HBase Security Configuration](#).
- The `hbase-site.xml` file, which must be configured to use Kerberos security, must be in Flume's classpath or `HBASE_HOME/conf`.
- HBaseSink `org.apache.flume.sink.hbase.HBaseSink` supports secure HBase, but AsyncHBaseSink `org.apache.flume.sink.hbase.AsyncHBaseSink` does not.
- The Flume HBase sink takes the `kerberosPrincipal` and `kerberosKeytab` parameters:
  - `kerberosPrincipal` – specifies the Kerberos principal to be used
  - `kerberosKeytab` – specifies the path to the Kerberos keytab
  - These are defined as:

```
agent.sinks.hbaseSink.kerberosPrincipal = flume/fully.qualified.domain.name@YOUR-REALM.COM
agent.sinks.hbaseSink.kerberosKeytab = /etc/flume-ng/conf/flume.keytab
```

  - You can use the $KERBEROS_PRINCIPAL and $KERBEROS_KEYTAB substitution variables to configure the principal name and the keytab file path. See the following documentation for steps on how to configure the substitution variables: [Use Substitution Variables for the Kerberos Principal and Keytab](#).

- If HBase is running with the AccessController coprocessor, the `flume` user (or whichever user the agent is running as) must have permissions to write to the same table and the column family that the sink is configured to write to. You can grant permissions using the `grant` command from HBase shell as explained in [HBase Security Configuration](#).
- The Flume HBase Sink does not currently support impersonation; it will write to HBase as the user the agent is being run as.
- If you want to use HDFS Sink and HBase Sink to write to HDFS and HBase from the same agent respectively, both sinks have to use the same principal and keytab. If you want to use different credentials, the sinks have to be on different agents.
- Each Flume agent machine that writes to HBase (using a configured HBase sink) needs a Kerberos principal of the form:

```
flume/fully.qualified.domain.name@YOUR-REALM.COM
```

where `fully.qualified.domain.name` is the fully qualified domain name of the given Flume agent host machine, and YOUR-REALM.COM is the Kerberos realm.

HBase Authentication

> **Warning:** Disabling security on a production HBase system is difficult and could cause data loss. Contact Cloudera Support if you need to disable security in HBase.

To configure HBase security, complete the following tasks:

1. **Configure HBase Authentication**: You must establish a mechanism for HBase servers and clients to securely identify themselves with HDFS, ZooKeeper, and each other. This ensures that hosts are who they claim to be.

   > **Note:**
   >
   > - To enable HBase to work with Kerberos security, you must perform the installation and configuration steps in Configuring Hadoop Security in CDH 5 and ZooKeeper Security Configuration.
   > - Although an HBase Thrift server can connect to a secured Hadoop cluster, access is not secured from clients to the HBase Thrift server. To encrypt communication between clients and the HBase Thrift Server, see Configuring TLS/SSL for HBase Thrift Server on page 202.

   The following sections describe how to use Apache HBase and CDH 5 with Kerberos security:

   - Configuring Kerberos Authentication for HBase on page 118
   - Configuring Secure HBase Replication on page 124
   - Configuring the HBase Client TGT Renewal Period on page 124

2. **Configure HBase Authorization**: You must establish rules for the resources that clients are allowed to access. For more information, see Configuring HBase Authorization on page 433.

Using the Hue HBase App

Hue includes an HBase App that allows you to interact with HBase through a Thrift proxy server. Because Hue sits between the Thrift server and the client, the Thrift server assumes that all HBase operations come from the hue user and not the client. To ensure that users in Hue are only allowed to perform HBase operations assigned to their own credentials, and not those of the hue user, you must enable HBase impersonation.

Configuring Kerberos Authentication for HBase

Using Kerberos for authentication for the HBase component requires that you also use Kerberos authentication for ZooKeeper. This means that HBase Master, RegionServer, and client hosts must each have a Kerberos principal for authenticating to the ZooKeeper ensemble. The steps below provide the details. Before you start, be sure that:

- Kerberos is enabled for the cluster, as detailed in Enabling Kerberos Authentication Using the Wizard.
- Kerberos principals for Cloudera Manager Server, HBase, and ZooKeeper hosts exist and are available for use. See Managing Kerberos Credentials Using Cloudera Manager on page 63 for details.

Cloudera Manager automatically configures authentication between HBase to ZooKeeper and sets up the HBase Thrift gateway to support impersonation (doAs). However, you must manually configure the HBase REST service for Kerberos (it currently uses Simple authentication by default, instead of Kerberos). See Configure HBase REST Server for Kerberos Authentication on page 119 for details.

> **Note:** Impersonation (doAs) cannot be used with Thrift framed transport (TFramedTransport) because SASL does not work with Thrift framed transport.

You can use either Cloudera Manager or the command line to configure Kerberos authentication for HBase. Using Cloudera Manager simplifies the process, but both approaches are detailed below. This page includes these topics:

Configuring Kerberos Authentication for HBase Using Cloudera Manager

Cloudera Manager simplifies the task of configuring Kerberos authentication for HBase.

Configure HBase Servers to Authenticate with a Secure HDFS Cluster Using Cloudera Manager

Required Role: Cluster Administrator or **Full Administrator**

1. Log on to Cloudera Manager Admin Console.
2. Go to the HBase service (select **Clusters** > **HBASE**).
3. Click the **Configuration** tab.
4. Under the Scope filter, click **HBase (Service-Wide)**.
5. Under the Category filter, click **Security**.
6. Ensure the Kerberos principal for the HBase service was generated.
7. Find the **HBase Secure Authentication** property (type "HBase Secure" in the Search box, if necessary), and confirm (or enter) the principal to use for HBase.
8. Select **kerberos** as the authentication type.
9. Click **Save Changes**.
10. Restart the role.
11. Restart the service. Select **Restart** from the **Actions** drop-down menu adjacent to HBASE-n (Cluster).

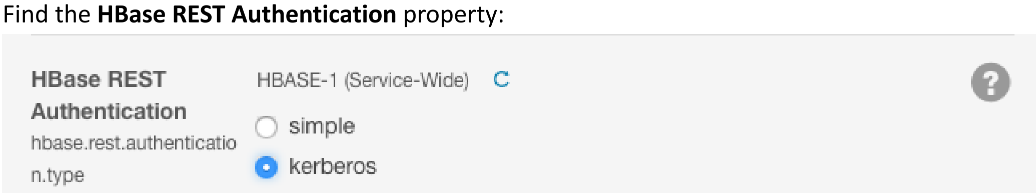
Configure HBase Servers and Clients to Authenticate with a Secure ZooKeeper

As mentioned above, secure HBase also requires secure ZooKeeper. The various HBase host systems—Master, RegionServer, and client—must have a principal to use to authenticate to the secure ZooKeeper ensemble. This is handled transparently by Cloudera Manager when you enable Kerberos as detailed above.

Configure HBase REST Server for Kerberos Authentication

Currently, the HBase REST Server uses Simple (rather than Kerberos) authentication by default. You must manually modify the setting using the Cloudera Manager Admin Console, as follows:

1. Log on to Cloudera Manager Admin Console.
2. Select **Clusters** > **HBASE**.
3. Click the **Configuration** tab.
4. Under the Scope filter, click **HBase (Service-Wide)**.
5. Under the Category filter, click **Security**.
6. Find the **HBase REST Authentication** property:



7. Click **kerberos** to select Kerberos instead of simple authentication.
8. Click **Save Changes**.
9. Restart the role.
10. Restart the service. Select **Restart** from the **Actions** drop-down menu adjacent to HBASE-n (Cluster).

Configuring Kerberos Authentication for HBase Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

Configure HBase Servers to Authenticate with a Secure HDFS Cluster Using the Command Line

To configure HBase servers to authenticate with a secure HDFS cluster, do the following:

Enable HBase Authentication

Set the `hbase.security.authentication` property to `kerberos` in `hbase-site.xml` on every host acting as an HBase master, RegionServer, or client. In CDH 5, `hbase.rpc.engine` is automatically detected and does not need to be set.

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

Configure HBase Kerberos Principals

To run on a secure HDFS cluster, HBase must authenticate itself to the HDFS services. HBase acts as a Kerberos principal and needs Kerberos credentials to interact with the Kerberos-enabled HDFS daemons. You can authenticate a service by using a keytab file, which contains a key that allows the service to authenticate to the Kerberos Key Distribution Center (KDC).

1. Create a service principal for the HBase server using the following syntax. This principal is used to authenticate the HBase server with the HDFS services. Cloudera recommends using `hbase` as the username.

```
$ kadmin
kadmin: addprinc -randkey hbase/fully.qualified.domain.name@YOUR-REALM.COM
```

   `fully.qualified.domain.name` is the host where the HBase server is running, and `YOUR-REALM` is the name of your Kerberos realm.

2. Create a keytab file for the HBase server.

```
$ kadmin
kadmin: xst -k hbase.keytab hbase/fully.qualified.domain.name
```

3. Copy the `hbase.keytab` file to the `/etc/hbase/conf` directory on the HBase server host. The owner of the `hbase.keytab` file should be the `hbase` user, and the file should have owner-only read permissions—that is, assign the file `0400` permissions and make it owned by `hbase:hbase`.

```
-r--------  1 hbase   hbase     1343 2012-01-09 10:39  hbase.keytab
```

4. To test that the keytab file was created properly, try to obtain Kerberos credentials as the HBase principal using only the keytab file. Substitute your `fully.qualified.domain.name` and realm in the following command:

```
$ kinit -k -t /etc/hbase/conf/hbase.keytab
hbase/fully.qualified.domain.name@YOUR-REALM.COM
```

5. In the `/etc/hbase/conf/hbase-site.xml` configuration file on *all* cluster hosts running the HBase daemon, add the following lines:

```
<property>
  <name>hbase.regionserver.kerberos.principal</name>
  <value>hbase/_HOST@YOUR-REALM.COM</value>
</property>

<property>
  <name>hbase.regionserver.keytab.file</name>
  <value>/etc/hbase/conf/hbase.keytab</value>
</property>

<property>
  <name>hbase.master.kerberos.principal</name>
  <value>hbase/_HOST@YOUR-REALM.COM</value>
```

```
</property>

<property>
<name>hbase.master.keytab.file</name>
<value>/etc/hbase/conf/hbase.keytab</value>
</property>
```

Configure HBase Servers and Clients to Authenticate with a Secure ZooKeeper

To run a secure HBase, you must also use a secure ZooKeeper. To use a secure ZooKeeper, each HBase host machine (Master, RegionServer, and client) must have a principal that allows it to authenticate with your secure ZooKeeper ensemble. The steps below assume that:

- ZooKeeper has been secured per the steps in [ZooKeeper Security Configuration](#).
- ZooKeeper is *not* managed by HBase.
- You have successfully completed steps above ([Enable HBase Authentication](#), [Configure HBase Kerberos Principals](#)) and have principal and keytab files in place for every HBase server and client.

To configure HBase Servers and clients to authenticate to ZooKeeper, you must:

Configure HBase JVMs (all Masters, RegionServers, and clients) to Use JAAS

1. On each host, set up a Java Authentication and Authorization Service (JAAS) by creating a `/etc/hbase/conf/zk-jaas.conf` file that contains the following:

```
   Client {
      com.sun.security.auth.module.Krb5LoginModule required
      useKeyTab=true
      useTicketCache=false
      keyTab="/etc/hbase/conf/hbase.keytab"
      principal="hbase/fully.qualified.domain.name@<YOUR-REALM>";
   };
```

2. Modify the `hbase-env.sh` file on HBase server and client hosts to include the following:

```
   export HBASE_OPTS="$HBASE_OPTS
-Djava.security.auth.login.config=/etc/hbase/conf/zk-jaas.conf"
   export HBASE_MANAGES_ZK=false
```

3. Restart the HBase cluster.

Configure the HBase Servers (Masters and RegionServers) to Use Authentication to Connect to ZooKeeper

> **Note:** These steps are required for command-line configuration only. Cloudera Manager does this automatically.

1. Update your `hbase-site.xml` on each HBase server host with the following properties:

```
<configuration>
   <property>
      <name>hbase.zookeeper.quorum</name>
      <value>$ZK_NODES</value>
   </property>
   <property>
      <name>hbase.cluster.distributed</name>
      <value>true</value>
   </property>
</configuration>
```

`$ZK_NODES` is the comma-separated list of hostnames of the ZooKeeper Quorum hosts that you configured according to the instructions in [ZooKeeper Security Configuration](#).

2. Add the following lines to the ZooKeeper configuration file `zoo.cfg`:

```
kerberos.removeHostFromPrincipal=true
kerberos.removeRealmFromPrincipal=true
```

3. Restart ZooKeeper.

Configure Authentication for the HBase REST and Thrift Gateways

By default, the REST gateway does not support impersonation, but accesses HBase as a statically configured user. The actual user who initiated the request is not tracked. With impersonation, the REST gateway user is a proxy user. The HBase server records the actual user who initiates each request and uses this information to apply authorization.

1. Enable support for proxy users by adding the following properties to `hbase-site.xml`. Substitute the REST gateway proxy user for *$USER*, and the allowed group list for *$GROUPS*.

```
<property>
  <name>hbase.security.authorization</name>
  <value>true</value>
</property>
<property>
  <name>hadoop.proxyuser.$USER.groups</name>
  <value>$GROUPS</value>
</property>
<property>
  <name>hadoop.proxyuser.$USER.hosts</name>
  <value>$GROUPS</value>
</property>
```

2. Enable REST gateway impersonation by adding the following to the `hbase-site.xml` file for every REST gateway:

```
<property>
  <name>hbase.rest.authentication.type</name>
  <value>kerberos</value>
</property>
<property>
  <name>hbase.rest.authentication.kerberos.principal</name>
  <value>HTTP/fully.qualified.domain.name@<YOUR-REALM/value>
</property>
<property>
  <name>hbase.rest.authentication.kerberos.keytab</name>
  <value>/etc/hbase/conf/hbase.keytab</value>
</property>
```

3. Add the following properties to `hbase-site.xml` for each Thrift gateway, replacing the Kerberos principal with a valid value:

```
<property>
  <name>hbase.thrift.keytab.file</name>
  <value>/etc/hbase/conf/hbase.keytab</value>
</property>
<property>
  <name>hbase.thrift.kerberos.principal</name>
  <value>hbase/fully.qualified.domain.name@<YOUR-REALM</value>
</property>
<property>
  <name>hbase.thrift.security.qop</name>
  <value>auth</value>
</property>
```

The value for the property `hbase.thrift.security.qop` can be one of the following:

- `auth-conf`—Authentication, integrity, and confidentiality checking
- `auth-int`—Authentication and integrity checking
- `auth`—Authentication checking only

4. To use the Thrift API principal to interact with HBase, add the `hbase.thrift.kerberos.principal` to the `acl` table. For example, to provide administrative access to the Thrift API principal `thrift_server`, run an HBase Shell command like the following:

```
hbase> grant'thrift_server', 'RWCA'
```

5. Optional: Configure HTTPS transport for Thrift by configuring the following parameters, substituting the placeholders with actual values:

```
<property>
  <name>hbase.thrift.ssl.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hbase.thrift.ssl.keystore.store</name>
  <value>LOCATION_OF_KEYSTORE</value>
</property>
<property>
  <name>hbase.thrift.ssl.keystore.password</name>
  <value>KEYSTORE_PASSWORD</value>
</property>
<property>
  <name>hbase.thrift.ssl.keystore.keypassword</name>
  <value>LOCATION_OF_KEYSTORE_KEY_PASSWORD</value>
</property>
```

The Thrift gateway authenticates with HBase using the supplied credential. No authentication is performed by the Thrift gateway itself. All client access through the Thrift gateway uses the gateway's credential, and all clients have its privileges.

Configure doAs Impersonation for the HBase Thrift Gateway

> **Note:** If you use framed transport, you cannot use doAs impersonation, because SASL does not work with Thrift framed transport.

doAs Impersonation provides a flexible way to use the same client to impersonate multiple principals. doAs is supported only in Thrift 1, not Thrift 2.

Enable doAs support by adding the following properties to `hbase-site.xml` on each Thrift gateway:

```
<property>
  <name>hbase.regionserver.thrift.http</name>
  <value>true</value>
</property>
<property>
  <name>hbase.thrift.support.proxyuser</name>
  <value>true/value>
</property>
```

See the demo client for information on using doAs impersonation in your client applications.

Start HBase

If the configuration worked, you see something similar to the following in the HBase Master and RegionServer logs when you start the cluster:

```
INFO zookeeper.ZooKeeper: Initiating client connection,
connectString=ZK_QUORUM_SERVER:2181 sessionTimeout=180000 watcher=master:60000
INFO zookeeper.ClientCnxn: Opening socket connection to server /ZK_QUORUM_SERVER:2181
INFO zookeeper.RecoverableZooKeeper: The identifier of this process is
PID@ZK_QUORUM_SERVER
INFO zookeeper.Login: successfully logged in.
INFO client.ZooKeeperSaslClient: Client will use GSSAPI as SASL mechanism.
```

```
INFO zookeeper.Login: TGT refresh thread started.
INFO zookeeper.ClientCnxn: Socket connection established to ZK_QUORUM_SERVER:2181,
initiating session
INFO zookeeper.Login: TGT valid starting at:        Sun Apr 08 22:43:59 UTC 2012
INFO zookeeper.Login: TGT expires:                  Mon Apr 09 22:43:59 UTC 2012
INFO zookeeper.Login: TGT refresh sleeping until: Mon Apr 09 18:30:37 UTC 2012
INFO zookeeper.ClientCnxn: Session establishment complete on server ZK_QUORUM_SERVER:2181,
 sessionid = 0x134106594320000, negotiated timeout = 180000
```

Configuring Secure HBase Replication

If you are using HBase Replication and you want to make it secure, read this section for instructions. Before proceeding, you should already have configured HBase Replication by following the instructions in the HBase Replication section of the *CDH 5 Installation Guide*.

To configure secure HBase replication, you must configure cross realm support for Kerberos, ZooKeeper, and Hadoop.

> **Note:** HBase peer-to-peer replication from a non-Kerberized cluster to a Kerberized cluster is not supported.

To configure secure HBase replication:

1. Create krbtgt principals for the two realms. For example, if you have two realms called `ONE.COM` and `TWO.COM`, you need to add the following principals: `krbtgt/ONE.COM@TWO.COM` and `krbtgt/TWO.COM@ONE.COM`. Add these two principals at both realms. There must be at least one common encryption mode between these two realms.

```
kadmin: addprinc -e "<enc_type_list>" krbtgt/ONE.COM@TWO.COM
kadmin: addprinc -e "<enc_type_list>" krbtgt/TWO.COM@ONE.COM
```

2. Add rules for creating short names in Zookeeper. To do this, add a system level property in `java.env`, defined in the `conf` directory. Here is an example rule that illustrates how to add support for the realm called `ONE.COM`, and have two members in the principal (such as `service/instance@ONE.COM`):

```
-Dzookeeper.security.auth_to_local=RULE:[2:\$1@\$0](.*@\\QONE.COM\\E$)s/@\\QONE.COM\\E$//DEFAULT
```

The above code example adds support for the `ONE.COM` realm in a different realm. So, in the case of replication, you must add a rule for the primary cluster realm in the replica cluster realm. `DEFAULT` is for defining the default rule.

3. Add rules for creating short names in the Hadoop processes. To do this, add the `hadoop.security.auth_to_local` property in the `core-site.xml` file in the replica cluster. For example, to add support for the `ONE.COM` realm:

```
<property>
  <name>hadoop.security.auth_to_local</name>
  <value>
    RULE:[2:$1@$0](.*@\QONE.COM\E$)s/@\QONE.COM\E$//
  DEFAULT
  </value>
</property>
```

For more information about adding rules, see Configuring the Mapping from Kerberos Principals to Short Names.

Configuring the HBase Client TGT Renewal Period

An HBase client user must also have a Kerberos principal which typically has a password that only the user knows. You should configure the `maxrenewlife` setting for the client's principal to a value that allows the user enough time to finish HBase client processes before the ticket granting ticket (TGT) expires. For example, if the HBase client processes require up to four days to complete, you should create the user's principal and configure the `maxrenewlife` setting by using this command:

```
kadmin: addprinc -maxrenewlife 4days
```

HCatalog Authentication

This section describes how to configure HCatalog in CDH 5 with Kerberos security in a Hadoop cluster:

- Before You Start on page 125
- Step 1: Create the HTTP keytab file on page 125
- Step 2: Configure WebHCat to Use Security on page 125
- Step 3: Create Proxy Users on page 126
- Step 4: Verify the Configuration on page 126

For more information about HCatalog see Installing and Using HCatalog.

Before You Start

Secure Web HCatalog requires a running remote Hive metastore service configured in secure mode. See Hive MetaStoreServer Security Configuration for instructions. Running secure WebHCat with an embedded repository is not supported.

Step 1: Create the HTTP keytab file

You need to create a `keytab` file for WebHCat. Follow these steps:

1. Create the file:

```
kadmin: addprinc -randkey HTTP/fully.qualified.domain.name@YOUR-REALM.COM
kadmin: xst -k HTTP.keytab HTTP/fully.qualified.domain.name
```

2. Move the file into the WebHCat configuration directory and restrict its access exclusively to the `hcatalog` user:

```
$ mv HTTP.keytab /etc/webhcat/conf/
$ chown hcatalog /etc/webhcat/conf/HTTP.keytab
$ chmod 400 /etc/webhcat/conf/HTTP.keytab
```

Step 2: Configure WebHCat to Use Security

Create or edit the WebHCat configuration file `webhcat-site.xml` in the configuration directory and set following properties:

Property	Value
templeton.kerberos.secret	Any random value
templeton.kerberos.keytab	`/etc/webhcat/conf/HTTP.keytab`
templeton.kerberos.principal	HTTP/fully.qualified.domain.name@YOUR-REALM.COM

Example configuration:

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

  <property>
    <name>templeton.kerberos.keytab</name>
    <value>/etc/webhcat/conf/HTTP.keytab</value>
```

```
  </property>

  <property>
    <name>templeton.kerberos.principal</name>
    <value>HTTP/fully.qualified.domain.name@YOUR-REALM.COM</value>
  </property>
```

Step 3: Create Proxy Users

WebHCat needs access to your NameNode to work properly, and so you must configure Hadoop to allow impersonation from the `hcatalog` user. To do this, edit your `core-site.xml` configuration file and set the `hadoop.proxyuser.HTTP.hosts` and `hadoop.proxyuser.HTTP.groups` properties to specify the hosts from which HCatalog can do the impersonation and what users can be impersonated. You can use the value * for "any".

Example configuration:

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

Step 4: Verify the Configuration

After restarting WebHcat you can verify that it is working by using `curl` (you may need to run `kinit` first):

```
$ curl --negotiate -i -u :
'http://fully.qualified.domain.name:50111/templeton/v1/ddl/database'
```

Hive Authentication

Hive authentication involves configuring Hive metastore, HiveServer2, and all Hive clients to use your deployment of LDAP/Active Directory Kerberos on your cluster.

Here is a summary of the status of Hive authentication in CDH 5:

- HiveServer2 supports authentication of the Thrift client using Kerberos or user/password validation backed by LDAP. For configuration instructions, see HiveServer2 Security Configuration.
- Earlier versions of HiveServer do not support Kerberos authentication for clients. However, the Hive MetaStoreServer does support Kerberos authentication for Thrift clients. For configuration instructions, see Hive MetaStoreServer Security Configuration.

  See also: Using Hive to Run Queries on a Secure HBase Server on page 133

For authorization, Hive uses Apache Sentry to enable role-based, fine-grained authorization for HiveServer2. See Apache Sentry Overview.

> **Important:** Cloudera does not support Apache Ranger or Hive's native authorization frameworks for configuring access control in Hive. Use Cloudera-supported Apache Sentry instead.

HiveServer2 Security Configuration

HiveServer2 supports authentication of the Thrift client using the following methods:

- Kerberos authentication
- LDAP authentication

  Starting with CDH 5.7, clusters running LDAP-enabled HiveServer2 deployments also accept Kerberos authentication. This ensures that users are not forced to enter usernames/passwords manually, and are able to take advantage of the multiple authentication schemes SASL offers. In CDH 5.6 and lower, HiveServer2 stops accepting delegation tokens when any alternate authentication is enabled.

Kerberos authentication is supported between the Thrift client and HiveServer2, and between HiveServer2 and secure HDFS. LDAP authentication is supported only between the Thrift client and HiveServer2.

To configure HiveServer2 to use one of these authentication modes, configure the `hive.server2.authentication` configuration property.

Enabling Kerberos Authentication for HiveServer2

If you configure HiveServer2 to use Kerberos authentication, HiveServer2 acquires a Kerberos ticket during startup. HiveServer2 requires a principal and keytab file specified in the configuration. Client applications (for example, JDBC or Beeline) must have a valid Kerberos ticket before initiating a connection to HiveServer2.

Configuring HiveServer2 for Kerberos-Secured Clusters

To enable Kerberos Authentication for HiveServer2, add the following properties in the `/etc/hive/conf/hive-site.xml` file:

```
<property>
  <name>hive.server2.authentication</name>
  <value>KERBEROS</value>
</property>
<property>
  <name>hive.server2.authentication.kerberos.principal</name>
  <value>hive/_HOST@YOUR-REALM.COM</value>
</property>
<property>
  <name>hive.server2.authentication.kerberos.keytab</name>
  <value>/etc/hive/conf/hive.keytab</value>
</property>
```

where:

- `hive.server2.authentication` is a client-facing property that controls the type of authentication HiveServer2 uses for connections to clients. In this case, HiveServer2 uses Kerberos to authenticate incoming clients.
- The `_HOST@YOUR-REALM.COM` value in the example above is the Kerberos principal for the host where HiveServer2 is running. The string _HOST in the properties is replaced at run time by the fully qualified domain name (FQDN) of the host machine where the daemon is running. Reverse DNS must be working on all the hosts configured this way. Replace `YOUR-REALM.COM` with the name of the Kerberos realm your Hadoop cluster is in.
- The `/etc/hive/conf/hive.keytab` value in the example above is a keytab file for that principal.

If you configure HiveServer2 to use both Kerberos authentication and secure impersonation, JDBC clients and Beeline can specify an alternate session user. If these clients have proxy user privileges, HiveServer2 impersonates the alternate user instead of the one connecting. The alternate user can be specified by the JDBC connection string `proxyUser=`*`userName`*

Configuring JDBC Clients for Kerberos Authentication with HiveServer2 (Using the Apache Hive Driver in Beeline)

JDBC-based clients must include `principal=<hive.server2.authentication.principal>` in the JDBC connection string. For example:

```
String url =
"jdbc:hive2://node1:10000/default;principal=hive/HiveServer2Host@YOUR-REALM.COM"
Connection con = DriverManager.getConnection(url);
```

where `hive` is the principal configured in `hive-site.xml` and `HiveServer2Host` is the host where HiveServer2 is running.

For JDBC clients using the **Cloudera JDBC driver**, see Cloudera JDBC Driver for Hive. For ODBC clients, see Cloudera ODBC Driver for Apache Hive.

Using Beeline to Connect to a Secure HiveServer2

Use the following command to start `beeline` and connect to a secure HiveServer2 process. In this example, the HiveServer2 process is running on `localhost` at port `10000`:

```
$ /usr/lib/hive/bin/beeline
beeline> !connect
jdbc:hive2://localhost:10000/default;principal=hive/HiveServer2Host@YOUR-REALM.COM
0: jdbc:hive2://localhost:10000/default>
```

For more information about the Beeline CLI, see Using the Beeline CLI.

For instructions on encrypting communication with the ODBC/JDBC drivers, see Configuring Encrypted Communication Between HiveServer2 and Client Drivers on page 204.

Using LDAP Username/Password Authentication with HiveServer2

As an alternative to Kerberos authentication, you can configure HiveServer2 to use user and password validation backed by LDAP. The client sends a username and password during connection initiation. HiveServer2 validates these credentials using an external LDAP service.

You can enable LDAP Authentication with HiveServer2 using Active Directory or OpenLDAP.

> **Important:** When using LDAP username/password authentication with HiveServer2, you must enable encrypted communication between HiveServer2 and its client drivers to avoid sending cleartext passwords. For instructions, see Configuring Encrypted Communication Between HiveServer2 and Client Drivers on page 204. To avoid sending LDAP credentials over a network in cleartext, see Configuring LDAPS Authentication with HiveServer2 on page 129.

Enabling LDAP Authentication with HiveServer2 using Active Directory

- **For managed clusters, use Cloudera Manager:**
  1. In the Cloudera Manager Admin Console, click **Hive** in the list of components, and then select the **Configuration** tab.
  2. Type "ldap" in the Search text box to locate the LDAP configuration fields.
  3. Check **Enable LDAP Authentication**.
  4. Enter the **LDAP URL** in the format `ldap[s]://<host>:<port>`
  5. Enter the **Active Directory Domain** for your environment.
  6. Click **Save Changes**.
- **For unmanaged clusters, use the command line:**

  Add the following properties to the `hive-site.xml`:

```
<property>
  <name>hive.server2.authentication</name>
  <value>LDAP</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.url</name>
  <value>LDAP_URLL</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.Domain</name>
  <value>AD_DOMAIN_ADDRESS</value>
</property>
```

Where:

The *LDAP_URL* value is the access URL for your LDAP server. For example, `ldap[s]://<host>:<port>`

Enabling LDAP Authentication with HiveServer2 using OpenLDAP

To enable LDAP authentication using OpenLDAP, include the following properties in `hive-site.xml`:

```
<property>
  <name>hive.server2.authentication</name>
  <value>LDAP</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.url</name>
  <value>LDAP_URL</value>
</property>
<property>
  <name>hive.server2.authentication.ldap.baseDN</name>
  <value>LDAP_BaseDN</value>
</property>
```

where:

- The `LDAP_URL` value is the access URL for your LDAP server.
- The `LDAP_BaseDN` value is the base LDAP DN for your LDAP server; for example, `ou=People,dc=example,dc=com`.

Configuring JDBC Clients for LDAP Authentication with HiveServer2

The JDBC client requires a connection URL as shown below.

JDBC-based clients must include `user=LDAP_Userid;password=LDAP_Password` in the JDBC connection string. For example:

```
String url = "jdbc:hive2://node1:10000/default;user=LDAP_Userid;password=LDAP_Password"
Connection con = DriverManager.getConnection(url);
```

where the `LDAP_Userid` value is the user ID and `LDAP_Password` is the password of the client user.

For ODBC Clients, see Cloudera ODBC Driver for Apache Hive.

Enabling LDAP Authentication for HiveServer2 in Hue

Enable LDAP authentication with HiveServer2 by setting the following properties under the `[beeswax]` section in `hue.ini`.

`auth_username`	LDAP username of Hue user to be authenticated.
`auth_password`	LDAP password of Hue user to be authenticated.

Hive uses these login details to authenticate to LDAP. The Hive service trusts that Hue has validated the user being impersonated.

Configuring LDAPS Authentication with HiveServer2

HiveServer2 supports LDAP username/password authentication for clients. Clients send LDAP credentials to HiveServer2 which in turn verifies them against the configured LDAP provider, such as OpenLDAP or Microsoft Active Directory. Most implementations now support LDAPS (LDAP over TLS/SSL), an authentication protocol that uses TLS/SSL to encrypt communication between the LDAP service and its client (in this case, HiveServer2) to avoid sending LDAP credentials in cleartext.

To configure the LDAPS service with HiveServer2:

1. Import the LDAP server CA certificate or the server certificate into a truststore on the HiveServer2 host. If you import the CA certificate, HiveServer2 will trust any server with a certificate issued by the LDAP server's CA. If you only import the server certificate, HiveServer2 trusts only that server. See Understanding Keystores and Truststores on page 181 for more details.
2. Make sure the truststore file is readable by the `hive` user.

3. Set the `hive.server2.authentication.ldap.url` configuration property in `hive-site.xml` to the LDAPS URL. For example, `ldaps://sample.myhost.com`.

   > **Note:** The URL scheme should be `ldaps` and *not* `ldap`.

4. If this is a managed cluster, in Cloudera Manager, go to the Hive service and select **Configuration**. Under Category, select **Security**. In the right panel, search for **HiveServer2 TLS/SSL Certificate Trust Store File**, and add the path to the truststore file that you created in step 1.

   If you are using an unmanaged cluster, set the environment variable HADOOP_OPTS as follows:

```
HADOOP_OPTS="-Djavax.net.ssl.trustStore=<trustStore-file-path>
-Djavax.net.ssl.trustStorePassword=<trustStore-password>"
```

5. Restart HiveServer2.

Pluggable Authentication

Pluggable authentication allows you to provide a custom authentication provider for HiveServer2.

To enable pluggable authentication:

1. Set the following properties in `/etc/hive/conf/hive-site.xml`:

```
<property>
  <name>hive.server2.authentication</name>
  <value>CUSTOM</value>
  <description>Client authentication types.
  NONE: no authentication check
  LDAP: LDAP/AD based authentication
  KERBEROS: Kerberos/GSSAPI authentication
  CUSTOM: Custom authentication provider
  (Use with property hive.server2.custom.authentication.class)
  </description>
</property>

<property>
  <name>hive.server2.custom.authentication.class</name>
  <value>pluggable-auth-class-name</value>
  <description>
  Custom authentication class. Used when property
  'hive.server2.authentication' is set to 'CUSTOM'. Provided class
  must be a proper implementation of the interface
  org.apache.hive.service.auth.PasswdAuthenticationProvider. HiveServer2
  will call its Authenticate(user, passed) method to authenticate requests.
  The implementation may optionally extend the Hadoop's
  org.apache.hadoop.conf.Configured class to grab Hive's Configuration object.
  </description>
</property>
```

2. Make the class available in the CLASSPATH of HiveServer2.

Trusted Delegation with HiveServer2

HiveServer2 determines the identity of the connecting user from the authentication subsystem (Kerberos or LDAP). Any new session started for this connection runs on behalf of this connecting user. If the server is configured to proxy the user at the Hadoop level, then all MapReduce jobs and HDFS accesses will be performed with the identity of the connecting user. If Apache Sentry is configured, then this connecting userid can also be used to verify access rights to underlying tables and views.

Users with Hadoop superuser privileges can request an alternate user for the given session. HiveServer2 checks that the connecting user can proxy the requested userid, and if so, runs the new session as the alternate user. For example, the Hadoop superuser hue can request that a connection's session be run as user bob.

Alternate users for new JDBC client connections are specified by adding the `hive.server2.proxy.user=`*`alternate_user_id`* property to the JDBC connection URL. For example, a JDBC connection string that lets user hue run a session as user bob would be as follows:

```
# Login as super user Hue
kinit hue -k -t hue.keytab hue@MY-REALM.COM

# Connect using following JDBC connection string
#
jdbc:hive2://myHost.myOrg.com:10000/default;principal=hive/_HOST@MY-REALM.COM;hive.server2.proxy.user=bob
```

The connecting user must have Hadoop-level proxy privileges over the alternate user.

HiveServer2 Impersonation

> **Important:** This is not the recommended method to implement HiveServer2 authorization. Cloudera recommends you use Sentry to implement this instead.

Impersonation in HiveServer2 allows users to execute queries and access HDFS files as the connected user rather than the super user who started the HiveServer2 daemon. This enforces an access control policy at the file level using HDFS file permissions or ACLs. Keeping impersonation enabled means Sentry does not have end-to-end control over the authorization process. While Sentry can enforce access control policies on tables and views in the Hive warehouse, it has no control over permissions on the underlying table files in HDFS. Hence, even if users do not have the Sentry privileges required to access a table in the warehouse, as long as they have permission to access the corresponding table file in HDFS, any jobs or queries submitted will bypass Sentry authorization checks and execute successfully.

To configure Sentry correctly, restrict ownership of the Hive warehouse to `hive:hive` and disable Hive impersonation as described here.

To enable impersonation in HiveServer2:

1. Add the following property to the `/etc/hive/conf/hive-site.xml` file and set the value to `true`. (The default value is `false`.)

```
<property>
  <name>hive.server2.enable.impersonation</name>
  <description>Enable user impersonation for HiveServer2</description>
  <value>true</value>
</property>
```

2. In HDFS or MapReduce configurations, add the following property to the `core-site.xml` file:

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

See also File System Permissions.

Securing the Hive Metastore

> **Note:** This is not the recommended method to protect the Hive Metastore. Cloudera recommends you use Sentry to implement this instead.

To prevent users from accessing the Hive metastore and the Hive metastore database using any method other than through HiveServer2, the following actions are recommended:

- Add a firewall rule on the metastore service host to allow access to the metastore port only from the HiveServer2 host. You can do this using iptables.
- Grant access to the metastore database only from the metastore service host. This is specified for MySQL as:

```
GRANT ALL PRIVILEGES ON metastore.* TO 'hive'@'metastorehost';
```

  where `metastorehost` is the host where the metastore service is running.

- Make sure users who are not admins cannot log on to the host on which HiveServer2 runs.

Disabling the Hive Security Configuration

Hive's security related metadata is stored in the configuration file `hive-site.xml`. The following sections describe how to disable security for the Hive service.

Disable Client/Server Authentication

To disable client/server authentication, set `hive.server2.authentication` to `NONE`. For example,

```
<property>
  <name>hive.server2.authentication</name>
  <value>NONE</value>
  <description>
    Client authentication types.
       NONE: no authentication check
       LDAP: LDAP/AD based authentication
       KERBEROS: Kerberos/GSSAPI authentication
       CUSTOM: Custom authentication provider
               (Use with property hive.server2.custom.authentication.class)
  </description>
</property>
```

Disable Hive Metastore security

To disable Hive Metastore security, perform the following steps:

- Set the `hive.metastore.sasl.enabled` property to `false` in all configurations, the metastore service side as well as for all clients of the metastore. For example, these might include HiveServer2, Impala, Pig and so on.
- Remove or comment the following parameters in `hive-site.xml` for the metastore service. Note that this is a server-only change.
  - `hive.metastore.kerberos.keytab.file`
  - `hive.metastore.kerberos.principal`

Disable Underlying Hadoop Security

If you also want to disable the underlying Hadoop security, remove or comment out the following parameters in `hive-site.xml`.

- `hive.server2.authentication.kerberos.keytab`
- `hive.server2.authentication.kerberos.principal`

Hive Metastore Server Security Configuration

> **Important:**
>
> This section describes how to configure security for the Hive metastore server. If you are using HiveServer2, see HiveServer2 Security Configuration.

Here is a summary of Hive metastore server security in CDH 5:

- No additional configuration is required to run Hive on top of a security-enabled Hadoop cluster in standalone mode using a local or embedded metastore.

- HiveServer does not support Kerberos authentication for clients. While it is possible to run HiveServer with a secured Hadoop cluster, doing so creates a security hole since HiveServer does not authenticate the Thrift clients that connect to it. Instead, you can use HiveServer2 HiveServer2 Security Configuration.
- The Hive metastore server supports Kerberos authentication for Thrift clients. For example, you can configure a standalone Hive metastore server instance to force clients to authenticate with Kerberos by setting the following properties in the `hive-site.xml` configuration file used by the metastore server:

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

<property>
  <name>hive.metastore.kerberos.keytab.file</name>
  <value>/etc/hive/conf/hive.keytab</value>
  <description>The path to the Kerberos Keytab file containing the metastore thrift
server's service principal.</description>
</property>

<property>
  <name>hive.metastore.kerberos.principal</name>
  <value>hive/_HOST@YOUR-REALM.COM</value>
  <description>The service principal for the metastore thrift server. The special string
 _HOST will be replaced automatically with the correct host name.</description>
</property>
```

Note:

The values shown above for the `hive.metastore.kerberos.keytab.file` and `hive.metastore.kerberos.principal` properties are examples which you will need to replace with the appropriate values for your cluster. Also note that the Hive keytab file should have its access permissions set to `600` and be owned by the same account that is used to run the Metastore server, which is the `hive` user by default.

- Requests to access the metadata are fulfilled by the Hive metastore impersonating the requesting user. This includes read access to the list of databases, tables, properties of each table such as their HDFS location and file type. You can restrict access to the Hive metastore service by allowing it to impersonate only a subset of Kerberos users. This can be done by setting the `hadoop.proxyuser.hive.groups` property in `core-site.xml` on the Hive metastore host.

  For example, if you want to give the `hive` user permission to impersonate members of groups `hive` and `user1`:

```
<property>
<name>hadoop.proxyuser.hive.groups</name>
<value>hive,user1</value>
</property>
```

  In this example, the Hive metastore can impersonate users belonging to *only* the `hive` and `user1` groups. Connection requests from users not belonging to these groups will be rejected.

Using Hive to Run Queries on a Secure HBase Server

To use Hive to run queries on a secure HBase Server, you must set the following `HIVE_OPTS` environment variable:

```
env HIVE_OPTS="-hiveconf hbase.security.authentication=kerberos -hiveconf
hbase.master.kerberos.principal=hbase/_HOST@YOUR-REALM.COM -hiveconf
hbase.regionserver.kerberos.principal=hbase/_HOST@YOUR-REALM.COM -hiveconf
hbase.zookeeper.quorum=zookeeper1,zookeeper2,zookeeper3" hive
```

where:

- You replace `YOUR-REALM` with the name of your Kerberos realm
- You replace `zookeeper1,zookeeper2,zookeeper3` with the names of your ZooKeeper servers. The `hbase.zookeeper.quorum` property is configured in the `hbase-site.xml` file.
- The special string `_HOST` is replaced at run-time by the fully qualified domain name of the host machine where the HBase Master or RegionServer is running. This requires that reverse DNS is properly working on all the hosts configured this way.

In the following, `_HOST` is the name of the host where the HBase Master is running:

```
-hiveconf hbase.master.kerberos.principal=hbase/_HOST@YOUR-REALM.COM
```

In the following, `_HOST` is the hostname of the HBase RegionServer that the application is connecting to:

```
-hiveconf hbase.regionserver.kerberos.principal=hbase/_HOST@YOUR-REALM.COM
```

Note:

You can also set the `HIVE_OPTS` environment variable in your shell profile.

HttpFS Authentication

This section describes how to configure HttpFS CDH 5 with Kerberos security on a Hadoop cluster:

- Configuring the HttpFS Server to Support Kerberos Security on page 134
- Using curl to access an URL Protected by Kerberos HTTP SPNEGO on page 136

For more information about HttpFS, see https://archive.cloudera.com/cdh5/cdh/5/hadoop/hadoop-hdfs-httpfs/index.html.

Important:

To enable HttpFS to work with Kerberos security on your Hadoop cluster, make sure you perform the installation and configuration steps in Configuring Hadoop Security in CDH 5.

Important:

If the NameNode, Secondary NameNode, DataNode, JobTracker, TaskTrackers, ResourceManager, NodeManagers, HttpFS, or Oozie services are configured to use Kerberos HTTP SPNEGO authentication, and two or more of these services are running on the same host, then all of the running services must use the same HTTP principal and keytab file used for their HTTP endpoints.

Configuring the HttpFS Server to Support Kerberos Security

1. Create an HttpFS service user principal that is used to authenticate with the Hadoop cluster. The syntax of the principal is: `httpfs/<fully.qualified.domain.name>@<YOUR-REALM>` where: `fully.qualified.domain.name` is the host where the HttpFS server is running `YOUR-REALM` is the name of your Kerberos realm

```
kadmin: addprinc -randkey httpfs/fully.qualified.domain.name@YOUR-REALM.COM
```

2. Create a HTTP service user principal that is used to authenticate user requests coming to the HttpFS HTTP web-services. The syntax of the principal is: `HTTP/<fully.qualified.domain.name>@<YOUR-REALM>` where:

`'fully.qualified.domain.name'` is the host where the HttpFS server is running `YOUR-REALM` is the name of your Kerberos realm

```
kadmin: addprinc -randkey HTTP/fully.qualified.domain.name@YOUR-REALM.COM
```

> **Important:**
>
> The HTTP/ component of the HTTP service user principal must be upper case as shown in the syntax and example above.

3. Create keytab files with both principals.

```
$ kadmin
kadmin: xst -k httpfs.keytab httpfs/fully.qualified.domain.name
kadmin: xst -k http.keytab HTTP/fully.qualified.domain.name
```

4. Merge the two keytab files into a single keytab file:

```
$ ktutil
ktutil: rkt httpfs.keytab
ktutil: rkt http.keytab
ktutil: wkt httpfs-http.keytab
```

5. Test that credentials in the merged keytab file work. For example:

```
$ klist -e -k -t httpfs-http.keytab
```

6. Copy the `httpfs-http.keytab` file to the HttpFS configuration directory. The owner of the `httpfs-http.keytab` file should be the `httpfs` user and the file should have owner-only read permissions.
7. Edit the HttpFS server `httpfs-site.xml` configuration file in the HttpFS configuration directory by setting the following properties:

Property	Value
httpfs.authentication.type	kerberos
httpfs.hadoop.authentication.type	kerberos
httpfs.authentication.kerberos.principal	HTTP/<HTTPFS-HOSTNAME>@<YOUR-REALM.COM>
httpfs.authentication.kerberos.keytab	/etc/hadoop-httpfs/conf/httpfs-http.keytab
httpfs.hadoop.authentication.kerberos.principal	httpfs/<HTTPFS-HOSTNAME>@<YOUR-REALM.COM>
httpfs.hadoop.authentication.kerberos.keytab	/etc/hadoop-httpfs/conf/httpfs-http.keytab
httpfs.authentication.kerberos.name.rules	Use the value configured for 'hadoop.security.auth_to_local' in 'core-site.xml'

> **Important:**
>
> You must restart the HttpFS server to have the configuration changes take effect.

Using curl to access an URL Protected by Kerberos HTTP SPNEGO

> **Important:**
>
> Your version of `curl` must support GSS and be capable of running `curl -V`.

To configure curl to access an URL protected by Kerberos HTTP SPNEGO:

1. Run `curl -V`:

```
$ curl -V
curl 7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8l
zlib/1.2.3
Protocols: tftp ftp telnet dict ldap http file https ftps
Features: GSS-Negotiate IPv6 Largefile NTLM SSL libz
```

2. Login to the KDC using `kinit`.

```
$ kinit
Please enter the password for tucu@LOCALHOST:
```

3. Use `curl` to fetch the protected URL:

```
$ curl --cacert
              /path/to/truststore.pem --negotiate -u : -b ~/cookiejar.txt -c
~/cookiejar.txt https://localhost:14000/webhdfs/v1/?op=liststatus
```

where:

- The `--cacert` option is required if you are using TLS/SSL certificates that curl does not recognize by default.
- The `--negotiate` option enables SPNEGO in `curl`.
- The `-u :` option is required but the username is ignored (the principal that has been specified for `kinit` is used).
- The `-b` and `-c` options are used to store and send HTTP cookies.
- Cloudera does not recommend using the `-k` or `--insecure` option as it turns off curl's ability to verify the certificate.

Hue Authentication

This page describes properties in the Hue configuration file, `hue.ini`, that support authentication and Hue security in general.

For information on configuring Hue with Kerberos, , encrypting session communication, and enabling single sign-on with SAML, see:

- Configuring Kerberos Authentication for Hue on page 138
- Authenticate Hue Users with LDAP and Synchronize Hue with LDAP Server
- Authenticate Hue Users with SAML
- Authorize Hue User Groups with Sentry

Enabling LDAP Authentication with HiveServer2 and Impala

LDAP authentication with HiveServer2 and Impala can be enabled by setting the following properties under their respective sections in `hue.ini`, `[beeswax]` and `[impala]`.

`auth_username`	LDAP username of Hue user to be authenticated.
`auth_password`	LDAP password of Hue user to be authenticated.

These login details are only used by Impala and Hive to authenticate to LDAP. The Impala and Hive services trust Hue to have already validated the user being impersonated, rather than simply passing on the credentials.

Securing Sessions

When a session expires, the screen blurs and the user is automatically logged out of the Hue Web UI. Logging on returns the user to same location.

Session Timeout

User sessions are controlled with the `ttl` (time-to-live) property under `[desktop]> [[session]]` in `hue.ini`. After n seconds, the session expires whether active or not.

`ttl`	The cookie with the users session ID expires after n seconds. Default: `ttl=1209600` which is 60*60*24*14 seconds or 2 weeks

Idle Session Timeout

Idle sessions are controlled with the idle_session_timeout property under `[desktop] > [[auth]]` in `hue.ini`. Sessions that are idle for n seconds, expire. You can disable this property by setting it to a negative value.

`idle_session_timeout`	User session IDs expire after idle for n seconds. A negative value means idle sessions do not expire. `idle_session_timeout=900` means that sessions expire after being idle for 15 minutes `idle_session_timeout=-1` means that idle sessions do not expire (until `ttl`)

Secure Login

Login properties are set in [`hue.ini`]() under `[desktop] > [[auth]]`. They are based on [django-axes 1.5.0]().

`change_default_password`	If true, users must change password on first login. Must enable `backend=desktop.auth.backend.AllowFirstUserDjangoBackend`
`expires_after`	User accounts are disabled n seconds after logout. If negative, user sessions never expire.
`expire_superusers`	Apply `expires_after` to superusers.
`login_cooloff_time`	Failed logins are forgotten after n seconds.
`login_failure_limit`	Number of login attempts allowed before a record is created for failed logins.
`login_lock_out_at_failure`	If true, lock out IP after exceeding `login_failure_limit`. If `login_lock_out_by_combination_user_and_ip=true`, lock out IP and user. If `login_lock_out_use_user_agent=true`, also lock out user agent.
`login_lock_out_by_combination_user_and_ip`	If true, lock out IP *and* user.
`login_lock_out_use_user_agent`	If true, lock out user agent (such as a browser).

Secure Cookies

Secure session cookies can be enabled by specifying the `secure` configuration property under the `[desktop]> [[session]]` section in `hue.ini`. Additionally, you can set the `http_only` flag for cookies containing users' session IDs.

`secure`	The cookie with the user session ID is secure. Should only be enabled with HTTPS. Default: `false`
`http_only`	The cookie with the user session ID uses the HTTP only flag. Default: `true` *If the HttpOnly flag is included in the HTTP response header, the cookie cannot be accessed through a client side script.*
`expire_at_browser_close`	Use session-length cookies. Logs out the user when the browser window is closed. Default: `false`

Allowed HTTP Methods

You can specify the HTTP request methods that the server should respond to using the `http_allowed_methods` property under the `[desktop]` section in `hue.ini`.

`http_allowed_methods`	Default: `options,get,head,post,put,delete,connect`

Restricting the Cipher List

Cipher list support with HTTPS can be restricted by specifying the `ssl_cipher_list` configuration property under the `[desktop]` section in `hue.ini`.

`ssl_cipher_list`	Default: `!aNULL:!eNULL:!LOW:!EXPORT:!SSLv2`

URL Redirect Whitelist

Restrict the domains or pages to which Hue can redirect users. The `redirect_whitelist` property can be found under the `[desktop]` section in `hue.ini`.

`redirect_whitelist`	For example, to restrict users to your local domain and FQDN, the following value can be used: `^\/.*$,^http:\/\/www.mydomain.com\/.*$`

Oozie Permissions

Access to the Oozie dashboard and editor can be individually controlled in the Hue Web UI under **User Admin** > **Groups**.

Groups Property in UI	Description
oozie.dashboard_jobs_access	Enable Oozie Dashboard read-only access for all jobs. Default: `true`
oozie.disable_editor_access	Disable Oozie Editor access. Default: `false`

Configuring Kerberos Authentication for Hue

To configure the Hue server to support Hadoop security using Kerberos:

1. Create a Hue user principal in the same realm as the Hadoop cluster of the form:

```
kadmin: addprinc -randkey hue/hue.server.fully.qualified.domain.name@YOUR-REALM.COM
```

where: hue is the principal the Hue server is running as, `hue.server.fully.qualified.domain.name` is the fully qualified domain name (FQDN) of your Hue server, `YOUR-REALM.COM` is the name of the Kerberos realm your Hadoop cluster is in

2. Create a keytab file for the Hue principal using the same procedure that you used to create the keytab for the `hdfs` or `mapred` principal for a specific host. You should name this file `hue.keytab` and put this keytab file in the directory `/etc/hue` on the machine running the Hue server. Like all keytab files, this file should have the most limited set of permissions possible. It should be owned by the user running the hue server (usually hue) and should have the permission `400`.
3. To test that the keytab file was created properly, try to obtain Kerberos credentials as the Hue principal using only the keytab file. Substitute your FQDN and realm in the following command:

```
$ kinit -k -t /etc/hue/hue.keytab
hue/hue.server.fully.qualified.domain.name@YOUR-REALM.COM
```

4. In the `/etc/hue/hue.ini` configuration file, add the following lines in the sections shown. Replace the `kinit_path` value, `/usr/kerberos/bin/kinit`, shown below with the correct path on the user's system.

```
[desktop]

 [[kerberos]]
 # Path to Hue's Kerberos keytab file
 hue_keytab=/etc/hue/hue.keytab
 # Kerberos principal name for Hue
 hue_principal=hue/FQDN@REALM
 # add kinit path for non root users
 kinit_path=/usr/kerberos/bin/kinit

[beeswax]
 # If Kerberos security is enabled, use fully qualified domain name (FQDN)
   ## hive_server_host=<FQDN of Hive Server>
 # Hive configuration directory, where hive-site.xml is located
   ## hive_conf_dir=/etc/hive/conf

[impala]
   ## server_host=localhost
 # The following property is required when impalad and Hue
 # are not running on the same host
   ## impala_principal=impala/impalad.hostname.domainname.com

[search]
 # URL of the Solr Server
   ## solr_url=http://localhost:8983/solr/
 # Requires FQDN in solr_url if enabled
   ## security_enabled=false

[hadoop]

 [[hdfs_clusters]]

 [[[default]]]
 # Enter the host and port on which you are running the Hadoop NameNode
 namenode_host=FQDN
 hdfs_port=8020
 http_port=50070
 security_enabled=true

 # Thrift plugin port for the name node
   ## thrift_port=10090

 # Configuration for YARN (MR2)
 # ------------------------------------------------------------------------
 [[yarn_clusters]]

 [[[default]]]
 # Enter the host on which you are running the ResourceManager
   ## resourcemanager_host=localhost
 # Change this if your YARN cluster is Kerberos-secured
   ## security_enabled=false
```

```
 # Thrift plug-in port for the JobTracker
   ## thrift_port=9290

[liboozie]
 # The URL where the Oozie service runs on. This is required in order for users to submit
 jobs.
   ## oozie_url=http://localhost:11000/oozie
 # Requires FQDN in oozie_url if enabled
   ## security_enabled=false
```

> **Important:**
>
> In the `/etc/hue/hue.ini` file, verify the following:
>
> — Make sure the `jobtracker_host` property is set to the fully qualified domain name of the host running the JobTracker. The JobTracker hostname must be fully qualified in a secured environment.
>
> — Make sure the `fs.defaultfs` property under each [[hdfs_clusters]] section contains the fully-qualified domain name of the file system access point, which is typically the NameNode.
>
> — Make sure the `hive_conf_dir` property under the `[beeswax]` section points to a directory containing a valid `hive-site.xml` (either the original or a synced copy).
>
> — Make sure the FQDN specified for HiveServer2 is the same as the FQDN specified for the `hue_principal` configuration property. Without this, HiveServer2 will not work with security enabled.
>
> Also note that HiveServer2 currently does not support SSL when using Kerberos.

5. In the `/etc/hadoop/conf/core-site.xml` configuration file on all of your cluster nodes, add the following lines:

```
<!-- Hue security configuration -->
<property>
  <name>hue.kerberos.principal.shortname</name>
  <value>hue</value>
</property>
<property>
  <name>hadoop.proxyuser.hue.groups</name>
  <value>*</value> <!-- A group which all users of Hue belong to, or the wildcard value
 "*" -->
</property>
<property>
  <name>hadoop.proxyuser.hue.hosts</name>
  <value>hue.server.fully.qualified.domain.name</value>
</property>
```

> **Important:**
>
> Make sure you change the `/etc/hadoop/conf/core-site.xml` configuration file on *all* of your cluster nodes.

6. If Hue is configured to communicate to Hadoop using HttpFS, then you must add the following properties to `httpfs-site.xml`:

```
<property>
  <name>httpfs.proxyuser.hue.hosts</name>
  <value>fully.qualified.domain.name</value>
</property>
<property>
  <name>httpfs.proxyuser.hue.groups</name>
```

```
  <value>*</value>
</property>
```

7. Add the following properties to the Oozie server `oozie-site.xml` configuration file in the Oozie configuration directory:

```
<property>
   <name>oozie.service.ProxyUserService.proxyuser.hue.hosts</name>
   <value>*</value>
</property>
<property>
   <name>oozie.service.ProxyUserService.proxyuser.hue.groups</name>
   <value>*</value>
</property>
```

8. Restart the JobTracker to load the changes from the `core-site.xml` file.

```
$ sudo service hadoop-0.20-mapreduce-jobtracker restart
```

9. Restart Oozie to load the changes from the `oozie-site.xml` file.

```
$ sudo service oozie restart
```

10. Restart the NameNode, JobTracker, and all DataNodes to load the changes from the `core-site.xml` file.

```
$ sudo service hadoop-0.20-(namenode|jobtracker|datanode) restart
```

Impala Authentication

Authentication is the mechanism to ensure that only specified hosts and users can connect to Impala. It also verifies that when clients connect to Impala, they are connected to a legitimate server. This feature prevents spoofing such as ***impersonation*** (setting up a phony client system with the same account and group names as a legitimate user) and ***man-in-the-middle attacks*** (intercepting application requests before they reach Impala and eavesdropping on sensitive information in the requests or the results).

Impala supports authentication using either Kerberos or LDAP.

> **Note:** Regardless of the authentication mechanism used, Impala always creates HDFS directories and data files owned by the same user (typically `impala`). To implement user-level access to different databases, tables, columns, partitions, and so on, use the Sentry authorization feature, as explained in Enabling Sentry Authorization for Impala on page 415.

Once you are finished setting up authentication, move on to authorization, which involves specifying what databases, tables, HDFS directories, and so on can be accessed by particular users when they connect through Impala. See Enabling Sentry Authorization for Impala on page 415 for details.

Enabling Kerberos Authentication for Impala

Impala supports Kerberos authentication. For background information on enabling Kerberos authentication, see the topic on Configuring Hadoop Security in the CDH 5 Security Guide.

When using Impala in a managed environment, Cloudera Manager automatically completes Kerberos configuration. In an unmanaged environment, create a Kerberos principal for each host running `impalad` or `statestored`. Cloudera recommends using a consistent format, such as `impala/_HOST@Your-Realm`, but you can use any three-part Kerberos server principal.

In Impala 2.0 and later, `user()` returns the full Kerberos principal string, such as `user@example.com`, in a Kerberized environment.

> **Note:** Regardless of the authentication mechanism used, Impala always creates HDFS directories and data files owned by the same user (typically `impala`). To implement user-level access to different databases, tables, columns, partitions, and so on, use the Sentry authorization feature, as explained in Enabling Sentry Authorization for Impala on page 415.

An alternative form of authentication you can use is LDAP, described in Enabling LDAP Authentication for Impala on page 145.

Requirements for Using Impala with Kerberos

On version 5 of Red Hat Enterprise Linux and comparable distributions, some additional setup is needed for the `impala-shell` interpreter to connect to a Kerberos-enabled Impala cluster:

```
sudo yum install python-devel openssl-devel python-pip
sudo pip-python install ssl
```

> **Important:**
>
> - If you plan to use Impala in your cluster, you must configure your KDC to allow tickets to be renewed, and you must configure `krb5.conf` to request renewable tickets. Typically, you can do this by adding the `max_renewable_life` setting to your realm in `kdc.conf`, and by adding the `renew_lifetime` parameter to the `libdefaults` section of `krb5.conf`.
>
>   For more information about renewable tickets, see the Kerberos documentation.
> - The Impala Web UI does not support Kerberos authentication.
> - You cannot use the Impala resource management feature on a cluster that has Kerberos authentication enabled.

Start all `impalad` and `statestored` daemons with the `--principal` and `--keytab-file` flags set to the principal and full path name of the `keytab` file containing the credentials for the principal.

Impala supports the Cloudera ODBC driver and the Kerberos interface provided. To use Kerberos through the ODBC driver, the host type must be set depending on the level of the ODBD driver:

- `SecImpala` for the ODBC 1.0 driver.
- `SecBeeswax` for the ODBC 1.2 driver.
- Blank for the ODBC 2.0 driver or higher, when connecting to a secure cluster.
- `HS2NoSasl` for the ODBC 2.0 driver or higher, when connecting to a non-secure cluster.

To enable Kerberos in the Impala shell, start the `impala-shell` command using the `-k` flag.

To enable Impala to work with Kerberos security on your Hadoop cluster, make sure you perform the installation and configuration steps in Authentication in the CDH 5 Security Guide.

Configuring Impala to Support Kerberos Security

Enabling Kerberos authentication for Impala involves steps that can be summarized as follows:

- Creating service principals for Impala and the HTTP service. Principal names take the form: *`serviceName/fully.qualified.domain.name@KERBEROS.REALM`*
- Creating, merging, and distributing key tab files for these principals.
- Editing `/etc/default/impala` (in cluster not managed by Cloudera Manager), or editing the **Security** settings in the Cloudera Manager interface, to accommodate Kerberos authentication.

Enabling Kerberos for Impala

1. Create an Impala service principal, specifying the name of the OS user that the Impala daemons run under, the fully qualified domain name of each node running `impalad`, and the realm name. For example:

```
$ kadmin
kadmin: addprinc -requires_preauth -randkey
impala/impala_host.example.com@TEST.EXAMPLE.COM
```

2. Create an HTTP service principal. For example:

```
kadmin: addprinc -randkey HTTP/impala_host.example.com@TEST.EXAMPLE.COM
```

> **Note:** The HTTP component of the service principal must be uppercase as shown in the preceding example.

3. Create `keytab` files with both principals. For example:

```
kadmin: xst -k impala.keytab impala/impala_host.example.com
kadmin: xst -k http.keytab HTTP/impala_host.example.com
kadmin: quit
```

4. Use `ktutil` to read the contents of the two keytab files and then write those contents to a new file. For example:

```
$ ktutil
ktutil: rkt impala.keytab
ktutil: rkt http.keytab
ktutil: wkt impala-http.keytab
ktutil: quit
```

5. (Optional) Test that credentials in the merged keytab file are valid, and that the "renew until" date is in the future. For example:

```
$ klist -e -k -t impala-http.keytab
```

6. Copy the `impala-http.keytab` file to the Impala configuration directory. Change the permissions to be only read for the file owner and change the file owner to the `impala` user. By default, the Impala user and group are both named `impala`. For example:

```
$ cp impala-http.keytab /etc/impala/conf
$ cd /etc/impala/conf
$ chmod 400 impala-http.keytab
$ chown impala:impala impala-http.keytab
```

7. Add Kerberos options to the Impala defaults file, `/etc/default/impala`. Add the options for both the `impalad` and `statestored` daemons, using the IMPALA_SERVER_ARGS and IMPALA_STATE_STORE_ARGS variables. For example, you might add:

```
-kerberos_reinit_interval=60
-principal=impala_1/impala_host.example.com@TEST.EXAMPLE.COM
-keytab_file=/var/run/cloudera-scm-agent/process/3212-impala-IMPALAD/impala.keytab
```

For more information on changing the Impala defaults specified in `/etc/default/impala`, see [Modifying Impala Startup Options](#).

> **Note:** Restart `impalad` and `statestored` for these configuration changes to take effect.

Enabling Kerberos for Impala with a Proxy Server

A common configuration for Impala with High Availability is to use a proxy server to submit requests to the actual `impalad` daemons on different hosts in the cluster. This configuration avoids connection problems in case of machine failure, because the proxy server can route new requests through one of the remaining hosts in the cluster. This configuration also helps with load balancing, because the additional overhead of being the "coordinator node" for each query is spread across multiple hosts.

Although you can set up a proxy server with or without Kerberos authentication, typically users set up a secure Kerberized configuration. For information about setting up a proxy server for Impala, including Kerberos-specific steps, see Using Impala through a Proxy for High Availability.

Enabling Impala Delegation for Kerberos Users

See Configuring Impala Delegation for Hue and BI Tools on page 147 for details about the delegation feature that lets certain users submit queries using the credentials of other users.

Using TLS/SSL with Business Intelligence Tools

You can use Kerberos authentication, TLS/SSL encryption, or both to secure connections from JDBC and ODBC applications to Impala. See Configuring Impala to Work with JDBC and Configuring Impala to Work with ODBC for details.

Prior to CDH 5.7 / Impala 2.5, the Hive JDBC driver did not support connections that use both Kerberos authentication and SSL encryption. If your cluster is running an older release that has this restriction, to use both of these security features with Impala through a JDBC application, use the Cloudera JDBC Connector as the JDBC driver.

Enabling Access to Internal Impala APIs for Kerberos Users

For applications that need direct access to Impala APIs, without going through the HiveServer2 or Beeswax interfaces, you can specify a list of Kerberos users who are allowed to call those APIs. By default, the `impala` and `hdfs` users are the only ones authorized for this kind of access. Any users not explicitly authorized through the `internal_principals_whitelist` configuration setting are blocked from accessing the APIs. This setting applies to all the Impala-related daemons, although currently it is primarily used for HDFS to control the behavior of the catalog server.

Mapping Kerberos Principals to Short Names for Impala

In and higher, Impala recognizes the `auth_to_local` setting, specified through the HDFS configuration setting `hadoop.security.auth_to_local` or the Cloudera Manager setting **Additional Rules to Map Kerberos Principals to Short Names**. This feature is disabled by default, to avoid an unexpected change in security-related behavior. To enable it:

- For clusters not managed by Cloudera Manager, specify `--load_auth_to_local_rules=true` in the `impalad` and `catalogd`configuration settings.
- For clusters managed by Cloudera Manager, select the **Use HDFS Rules to Map Kerberos Principals to Short Names** checkbox to enable the service-wide `load_auth_to_local_rules` configuration setting. Then restart the Impala service.

See Using Auth-to-Local Rules to Isolate Cluster Users for general information about this feature.

Kerberos-Related Memory Overhead for Large Clusters

On a kerberized cluster with high memory utilization, `kinit` commands executed after every `'kerberos_reinit_interval'` may cause out-of-memory errors, because executing the command involves a fork of the Impala process. The error looks similar to the following:

```
Failed to obtain Kerberos ticket for principal: <varname>principal_details</varname>
Failed to execute shell cmd: 'kinit -k -t <varname>keytab_details</varname>',
error was: Error(12): Cannot allocate memory
```

The following command changes the `vm.overcommit_memory` setting immediately on a running host. However, this setting is reset when the host is restarted.

```
echo 1 > /proc/sys/vm/overcommit_memory
```

To change the setting in a persistent way, add the following line to the `/etc/sysctl.conf` file:

```
vm.overcommit_memory=1
```

Then run `sysctl -p`. No reboot is needed.

Enabling LDAP Authentication for Impala

Authentication is the process of allowing only specified named users to access the server (in this case, the Impala server). This feature is crucial for any production deployment, to prevent misuse, tampering, or excessive load on the server. Impala uses LDAP for authentication, verifying the credentials of each user who connects through `impala-shell`, Hue, a Business Intelligence tool, JDBC or ODBC application, and so on.

> **Note:** Regardless of the authentication mechanism used, Impala always creates HDFS directories and data files owned by the same user (typically `impala`). To implement user-level access to different databases, tables, columns, partitions, and so on, use the Sentry authorization feature, as explained in Enabling Sentry Authorization for Impala on page 415.

An alternative form of authentication you can use is Kerberos, described in Enabling Kerberos Authentication for Impala on page 141.

Requirements for Using Impala with LDAP

Authentication against LDAP servers is available in Impala 1.2.2 and higher. Impala 1.4.0 adds support for secure LDAP authentication through SSL and TLS.

The Impala LDAP support lets you use Impala with systems such as Active Directory that use LDAP behind the scenes.

Kerberos Authentication for Connections Between Impala Components

Only client->Impala connections can be authenticated by LDAP.

You must use the Kerberos authentication mechanism for connections between internal Impala components, such as between the `impalad`, `statestored`, and `catalogd` daemons. See Enabling Kerberos Authentication for Impala on page 141 on how to set up Kerberos for Impala.

Server-Side LDAP Setup

These requirements apply on the server side when configuring and starting Impala:

To enable LDAP authentication, set the following startup options for `impalad`:

- `--enable_ldap_auth` enables LDAP-based authentication between the client and Impala.
- `--ldap_uri` sets the URI of the LDAP server to use. Typically, the URI is prefixed with `ldap://`. In Impala 1.4.0 and higher, you can specify secure SSL-based LDAP transport by using the prefix `ldaps://`. The URI can optionally specify the port, for example: `ldap://ldap_server.cloudera.com:389` or `ldaps://ldap_server.cloudera.com:636`. (389 and 636 are the default ports for non-SSL and SSL LDAP connections, respectively.)
- For `ldaps://` connections secured by SSL, `--ldap_ca_certificate="/path/to/certificate/pem"` specifies the location of the certificate in standard `.PEM` format. Store this certificate on the local filesystem, in a location that only the `impala` user and other trusted users can read.

Support for Custom Bind Strings

When Impala connects to LDAP it issues a bind call to the LDAP server to authenticate as the connected user. Impala clients, including the Impala shell, provide the short name of the user to Impala. This is necessary so that Impala can use Sentry for role-based access, which uses short names.

However, LDAP servers often require more complex, structured usernames for authentication. Impala supports three ways of transforming the short name (for example, `'henry'`) to a more complicated string. If necessary, specify one of the following configuration options when starting the `impalad` daemon on each DataNode:

- `--ldap_domain`: Replaces the username with a string *`username@ldap_domain`*.
- `--ldap_baseDN`: Replaces the username with a “distinguished name” (DN) of the form: `uid=`*`userid`*`, ldap_baseDN`. (This is equivalent to a Hive option).
- `--ldap_bind_pattern`: This is the most general option, and replaces the username with the string *ldap_bind_pattern* where all instances of the string `#UID` are replaced with *userid*. For example, an `ldap_bind_pattern` of `"user=#UID,OU=foo,CN=bar"` with a username of `henry` will construct a bind name of `"user=henry,OU=foo,CN=bar"`.

For clusters not managed by Cloudera Manager, specify the option on the `impalad` command line. For clusters managed by Cloudera Manager 5.4.0 and higher, search for the configuration field names `ldap_domain`, `ldap_basedn`, or `ldap_bind_pattern`, fill in and save the appropriate field values, and restart the Impala service. Prior to Cloudera Manager 5.4.0, these values were filled in using the **Impala Daemon Command Line Argument Advanced Configuration Snippet (Safety Valve)** field.

These options are mutually exclusive; Impala does not start if more than one of these options is specified.

Secure LDAP Connections

To avoid sending credentials over the wire in cleartext, you must configure a secure connection between both the client and Impala, and between Impala and the LDAP server. The secure connection could use SSL or TLS.

Secure LDAP connections through SSL:

For SSL-enabled LDAP connections, specify a prefix of `ldaps://` instead of `ldap://`. Also, the default port for SSL-enabled LDAP connections is 636 instead of 389.

Secure LDAP connections through TLS:

TLS, the successor to the SSL protocol, is supported by most modern LDAP servers. Unlike SSL connections, TLS connections can be made on the same server port as non-TLS connections. To secure all connections using TLS, specify the following flags as startup options to the `impalad` daemon:

- `--ldap_tls` tells Impala to start a TLS connection to the LDAP server, and to fail authentication if it cannot be done.
- `--ldap_ca_certificate="`*`/path/to/certificate/pem`*`"` specifies the location of the certificate in standard `.PEM` format. Store this certificate on the local filesystem, in a location that only the `impala` user and other trusted users can read.

LDAP Authentication for impala-shell Interpreter

To connect to Impala using LDAP authentication, you specify command-line options to the `impala-shell` command interpreter and enter the password when prompted:

- `-l` enables LDAP authentication.
- `-u` sets the user. Per Active Directory, the user is the short username, not the full LDAP distinguished name. If your LDAP settings include a search base, use the `--ldap_bind_pattern` on the `impalad` daemon to translate the short user name from `impala-shell` automatically to the fully qualified name.
- `impala-shell` automatically prompts for the password.

For the full list of available `impala-shell` options, see impala-shell Configuration Options.

LDAP authentication for JDBC applications: See Configuring Impala to Work with JDBC for the format to use with the JDBC connection string for servers using LDAP authentication.

Enabling LDAP for Impala in Hue

Enabling LDAP for Impala in Hue Using Cloudera Manager

1. Go to the Hue service.
2. Click the Configuration tab.
3. Select **Scope** > **Hue Server**.
4. Select **Category** > **Advanced**.
5. Add the following properties to the **Hue Server Advanced Configuration Snippet (Safety Valve) for hue_safety_valve_server.ini** property.

```
[impala]
auth_username=<LDAP username of Hue user to be authenticated>
auth_password=<LDAP password of Hue user to be authenticated>
```

6. Click **Save Changes**.

Enabling LDAP for Impala in Hue Using the Command Line

LDAP authentication for the Impala app in Hue can be enabled by setting the following properties under the `[impala]` section in `hue.ini`.

`auth_username`	LDAP username of Hue user to be authenticated.
`auth_password`	LDAP password of Hue user to be authenticated.

These login details are only used by Impala to authenticate to LDAP. The Impala service trusts Hue to have already validated the user being impersonated, rather than simply passing on the credentials.

Enabling Impala Delegation for LDAP Users

See Configuring Impala Delegation for Hue and BI Tools on page 147 for details about the delegation feature that lets certain users submit queries using the credentials of other users.

LDAP Restrictions for Impala

The LDAP support is preliminary. It currently has only been tested against Active Directory.

Using Multiple Authentication Methods with Impala

Impala 2.0 and later automatically handles both Kerberos and LDAP authentication. Each `impalad` daemon can accept both Kerberos and LDAP requests through the same port. No special actions need to be taken if some users authenticate through Kerberos and some through LDAP.

Prior to Impala 2.0, you had to configure each `impalad` to listen on a specific port depending on the kind of authentication, then configure your network load balancer to forward each kind of request to a DataNode that was set up with the appropriate authentication type. Once the initial request was made using either Kerberos or LDAP authentication, Impala automatically handled the process of coordinating the work across multiple nodes and transmitting intermediate results back to the coordinator node.

Configuring Impala Delegation for Hue and BI Tools

When users submit Impala queries through a separate application, such as Hue or a business intelligence tool, typically all requests are treated as coming from the same user. In Impala 1.2 and higher, authentication is extended by a new feature that allows applications to pass along credentials for the users that connect to them (known as “delegation”), and issue Impala queries with the privileges for those users. Currently, the delegation feature is available only for Impala queries submitted through application interfaces such as Hue and BI tools; for example, Impala cannot issue queries using the privileges of the HDFS user.

The delegation feature is enabled by a startup option for `impalad`: `--authorized_proxy_user_config`. When you specify this option, users whose names you specify (such as hue) can delegate the execution of a query to another user. The query runs with the privileges of the delegated user, not the original user such as hue. The name of the delegated user is passed using the HiveServer2 configuration property `impala.doas.user`.

You can specify a list of users that the application user can delegate to, or * to allow a superuser to delegate to any other user. For example:

```
impalad --authorized_proxy_user_config 'hue=user1,user2;admin=*' ...
```

> **Note:** Make sure to use single quotes or escape characters to ensure that any * characters do not undergo wildcard expansion when specified in command-line arguments.

See Modifying Impala Startup Options for details about adding or changing `impalad` startup options. See this Cloudera blog post for background information about the delegation capability in HiveServer2.

To set up authentication for the delegated users:

- On the server side, configure either user/password authentication through LDAP, or Kerberos authentication, for all the delegated users. See Enabling LDAP Authentication for Impala on page 145 or Enabling Kerberos Authentication for Impala on page 141 for details.
- On the client side, follow the instructions in the "Using User Name and Password" section in the ODBC driver installation guide. Then search for "delegation" in that same installation guide to learn about the **Delegation UID** field and `DelegationUID` configuration keyword to enable the delegation feature for ODBC-based BI tools.

Llama Authentication

> **Note:**
>
> The use of the Llama component for integrated resource management within YARN is no longer supported with and higher. The Llama support code is removed entirely in and higher.
>
> For clusters running Impala alongside other data management components, you define static service pools to define the resources available to Impala and other components. Then within the area allocated for Impala, you can create dynamic service pools, each with its own settings for the Impala admission control feature.

This section describes how to configure Llama in CDH 5 with Kerberos security in a Hadoop cluster.

> **Note:** Llama has been tested only in a Cloudera Manager deployment. For information on using Cloudera Manager to configure Llama and Impala, see Installing Impala.

Configuring Llama to Support Kerberos Security

1. Create a Llama service user principal using the syntax: `llama/`*`fully.qualified.domain.name@YOUR-REALM`*. This principal is used to authenticate with the Hadoop cluster, where *fully.qualified.domain.name* is the host where Llama is running and *YOUR-REALM* is the name of your Kerberos realm:

```
$ kadmin
kadmin: addprinc -randkey
llama/fully.qualified.domain.name@YOUR-REALM
```

2. Create a keytab file with the Llama principal:

```
$ kadmin
kadmin: xst -k llama.keytab llama/fully.qualified.domain.name
```

3. Test that the credentials in the keytab file work. For example:

```
$ klist -e -k -t llama.keytab
```

4. Copy the `llama.keytab` file to the Llama configuration directory. The owner of the `llama.keytab` file should be the `llama` user and the file should have owner-only read permissions.
5. Edit the Llama `llama-site.xml` configuration file in the Llama configuration directory by setting the following properties:

Property	Value
`llama.am.server.thrift.security`	`true`
`llama.am.server.thrift.kerberos.keytab.file`	`llama/conf.keytab`
`llama.am.server.thrift.kerberos.server.principal.name`	`llama/`*`fully.qualified.domain.name`*
`llama.am.server.thrift.kerberos.notification.principal.name`	`impala`

6. Restart Llama to make the configuration changes take effect.

Oozie Authentication

This section describes how to configure Oozie CDH 5 with Kerberos security on a Hadoop cluster:

- Configuring Kerberos Authentication for the Oozie Server on page 149
- Configuring Oozie HA with Kerberos on page 151

Important:

To enable Oozie to work with Kerberos security on your Hadoop cluster, make sure you perform the installation and configuration steps in Configuring Hadoop Security in CDH 5. Also note that when Kerberos security is enabled in Oozie, a web browser that supports Kerberos HTTP SPNEGO is required to access the Oozie web-console (for example, Firefox, Internet Explorer or Chrome).

Important:

If the NameNode, Secondary NameNode, DataNode, JobTracker, TaskTrackers, ResourceManager, NodeManagers, HttpFS, or Oozie services are configured to use Kerberos HTTP SPNEGO authentication, and two or more of these services are running on the same host, then all of the running services must use the same HTTP principal and keytab file used for their HTTP endpoints.

Configuring Kerberos Authentication for the Oozie Server

1. Create a Oozie service user principal using the syntax: `oozie/<fully.qualified.domain.name>@<YOUR-REALM>`. This principal is used to authenticate with the Hadoop cluster. where: `fully.qualified.domain.name` is the host where the Oozie server is running `YOUR-REALM` is the name of your Kerberos realm.

```
kadmin: addprinc -randkey oozie/fully.qualified.domain.name@YOUR-REALM.COM
```

2. Create a HTTP service user principal using the syntax: `HTTP/<fully.qualified.domain.name>@<YOUR-REALM>`. This principal is used to authenticate user requests coming to the Oozie web-services. where: `fully.qualified.domain.name` is the host where the Oozie server is running `YOUR-REALM` is the name of your Kerberos realm.

```
kadmin: addprinc -randkey HTTP/fully.qualified.domain.name@YOUR-REALM.COM
```

> **Important:**
>
> The HTTP/ component of the HTTP service user principal must be upper case as shown in the syntax and example above.

3. Create keytab files with both principals.

```
$ kadmin
kadmin: xst -k oozie.keytab oozie/fully.qualified.domain.name
kadmin: xst -k http.keytab HTTP/fully.qualified.domain.name
```

4. Merge the two keytab files into a single keytab file:

```
$ ktutil
ktutil: rkt oozie.keytab
ktutil: rkt http.keytab
ktutil: wkt oozie-http.keytab
```

5. Test that credentials in the merged keytab file work. For example:

```
$ klist -e -k -t oozie-http.keytab
```

6. Copy the `oozie-http.keytab` file to the Oozie configuration directory. The owner of the `oozie-http.keytab` file should be the `oozie` user and the file should have owner-only read permissions.
7. Edit the Oozie server `oozie-site.xml` configuration file in the Oozie configuration directory by setting the following properties:

> **Important:** You must restart the Oozie server to have the configuration changes take effect.

Property	Value
`oozie.service.HadoopAccessorService.kerberos.enabled`	`true`
`local.realm`	`<REALM>`
`oozie.service.HadoopAccessorService.keytab.file`	`/etc/oozie/conf/oozie-http.keytab` for a package installation, or `<EXPANDED_DIR>/conf/oozie-http.keytab` for a tarball installation
`oozie.service.HadoopAccessorService.kerberos.principal`	`oozie/<fully.qualified.domain.name>@<YOUR-REALM.COM>`
`oozie.authentication.type`	`kerberos`
`oozie.authentication.kerberos.principal`	`HTTP/<fully.qualified.domain.name>@<YOUR-REALM.COM>`
`oozie.authentication.kerberos.name.rules`	Use the value configured for `hadoop.security.auth_to_local` in `core-site.xml`

Configuring Oozie HA with Kerberos

> **Important:**
>
> - If you use Cloudera Manager, do not use these command-line instructions. Use the Cloudera Manager Kerberos wizard instead, which automates the steps described in this section. If you have already enabled Kerberos, Cloudera Manager will automatically generate Kerberos credentials for the new Oozie server. It will also regenerate credentials for any existing servers.
> - This information applies specifically to CDH 5.13.0. If you use a lower version of CDH, see the documentation for that version located at Cloudera Documentation.

In CDH 5, you can configure multiple active Oozie servers against the same database, providing high availability for Oozie. For instructions on setting up Oozie HA, see Oozie High Availability

Let's assume a setup with three hosts running Oozie servers: `host1.example.com`, `host2.example.com`, and `host3.example.com`. The Load Balancer which directs traffic to the Oozie servers is running on `oozie.example.com`. Perform the following steps to configure Kerberos authentication on this Oozie HA-enabled deployment:

1. Assuming your Kerberos realm is `EXAMPLE.COM`, create the following Kerberos principals:
   - `oozie/host1.example.com@EXAMPLE.COM`
   - `oozie/host2.example.com@EXAMPLE.COM`
   - `oozie/host3.example.com@EXAMPLE.COM`
   - `HTTP/host1.example.com@EXAMPLE.COM`
   - `HTTP/host2.example.com@EXAMPLE.COM`
   - `HTTP/host3.example.com@EXAMPLE.COM`
   - For the Load Balancer: `HTTP/oozie.example.com@EXAMPLE.COM`
2. On each host, create a keytab file with the corresponding `oozie` and `HTTP` principals from the list above. Each keytab file should also have the Load Balancer's HTTP principal. For example, the keytab file on `host1` would comprise:
   - `oozie/host1.example.com@EXAMPLE.COM`
   - `HTTP/host1.example.com@EXAMPLE.COM`
   - `HTTP/oozie.example.com@EXAMPLE.COM`
3. On each host, configure the following properties in `oozie-site.xml`:

```
<property>
    <name>oozie.authentication.kerberos.principal</name>
    <value>HTTP/<hostname>@$EXAMPLE.COM</value>
    <description>
        Indicates the Kerberos principal to be used for HTTP endpoint.
       The principal MUST start with 'HTTP/' as per Kerberos HTTP SPNEGO specification.

    </description>
</property>

<property>
    <name>oozie.authentication.kerberos.keytab</name>
    <value>${oozie.service.HadoopAccessorService.keytab.file}</value>
    <description>
        Location of the keytab file with the credentials for the principal.
        Referring to the same keytab file Oozie uses for its Kerberos credentials for
Hadoop.
    </description>
</property>
```

Solr Authentication

This section describes how to configure Solr to enable authentication.

Authentication

When authentication is enabled, only specified hosts and users can connect to Solr. Authentication also verifies that clients connect to legitimate servers. This feature prevents spoofing such as impersonation and man-in-the-middle attacks. Search supports Kerberos and LDAP authentication.

Cloudera Search supports a variety of combinations of authentication protocols:

Table 5: Authentication Protocol Combinations

Solr Authentication	Use Case
No authentication	Insecure cluster
Kerberos only	The Hadoop cluster has Kerberos turned on and every user (or client) connecting to Solr has a Kerberos principal.
Kerberos and LDAP	The Hadoop cluster has Kerberos turned on. External Solr users (or clients) do not have Kerberos principals but do have identities in the LDAP server. Client authentication using LDAP requires that Kerberos is enabled for the cluster. Using LDAP alone is not supported.

Once you are finished setting up authentication, configure Sentry authorization. Authorization involves specifying which resources can be accessed by particular users when they connect through Search. See Configuring Sentry Authorization for Cloudera Search on page 425 for details.

Enabling Kerberos Authentication for Solr

Solr supports Kerberos authentication. All necessary packages are installed when you install Search. To enable Kerberos, create principals and keytabs and then modify default configurations.

> **Important:** The following instructions only apply to configuring Kerberos in an unmanaged environment. If you are using Cloudera Manager, Kerberos configuration is automatically handled. For more information on enabling Kerberos using Cloudera Manager, see Configuring Authentication in Cloudera Manager on page 40.

Create Principals and Keytabs

Repeat this process on all Solr server hosts:

1. Create a Solr service user principal using the syntax: `solr/solr01.example.com@EXAMPLE.COM`. This principal is used to authenticate with the Hadoop cluster. Replace `solr01.example.com` with the Solr server host, and `EXAMPLE.COM` with your Kerberos realm.

   ```
   $ kadmin
   kadmin: addprinc -randkey solr/solr01.example.com@EXAMPLE.COM
   ```

2. Create a HTTP service user principal using the syntax: `HTTP/solr01.example.com@EXAMPLE.COM`. This principal is used to authenticate user requests coming to the Solr web-services. Replace `solr01.example.com` with the Solr server host, and `EXAMPLE.COM` with your Kerberos realm.

   ```
   kadmin: addprinc -randkey HTTP/solr01.example.com@EXAMPLE.COM
   ```

   > **Note:**
   >
   > The `HTTP/` component of the HTTP service user principal must be upper case as shown in the syntax and example above.

3. Create keytab files with both principals.

   ```
   kadmin: xst -norandkey -k solr.keytab solr/solr01.example.com \
   HTTP/solr01.example.com
   ```

4. Test that credentials in the merged keytab file work. For example:

```
$ klist -e -k -t solr.keytab
```

5. Copy the `solr.keytab` file to the Solr configuration directory. The owner of the `solr.keytab` file should be the `solr` user and the file should have owner-only read permissions.

Modify Default Configurations

Repeat this process on all Solr server hosts:

1. Ensure that the following properties appear in `/etc/default/solr` or `/opt/cloudera/parcels/CDH-*/etc/default/solr` and that they are uncommented. Modify these properties to match your environment. The relevant properties to be uncommented and modified are:

```
SOLR_AUTHENTICATION_TYPE=kerberos
SOLR_AUTHENTICATION_SIMPLE_ALLOW_ANON=true
SOLR_AUTHENTICATION_KERBEROS_KEYTAB=/etc/solr/conf/solr.keytab
SOLR_AUTHENTICATION_KERBEROS_PRINCIPAL=HTTP/localhost@LOCALHOST
SOLR_AUTHENTICATION_KERBEROS_NAME_RULES=DEFAULT
SOLR_AUTHENTICATION_JAAS_CONF=/etc/solr/conf/jaas.conf
```

> **Note:** Modify the values for these properties to match your environment. For example, the `SOLR_AUTHENTICATION_KERBEROS_PRINCIPAL=HTTP/localhost@LOCALHOST` must include the principal instance and Kerberos realm for your environment. That is often different from `localhost@LOCALHOST`.

2. Set `hadoop.security.auth_to_local` to match the value specified by `SOLR_AUTHENTICATION_KERBEROS_NAME_RULES` in `/etc/default/solr` or `/opt/cloudera/parcels/CDH-*/etc/default/solr`.

> **Note:** For information on how to configure the rules, see Configuring the Mapping from Kerberos Principals to Short Names. For additional information on using Solr with HDFS, see Configuring Solr for Use with HDFS.

3. If using applications that use the `solrj` library, set up the Java Authentication and Authorization Service (JAAS).

   a. Create a `jaas.conf` file in the Solr configuration directory containing the following settings. This file and its location must match the `SOLR_AUTHENTICATION_JAAS_CONF` value. Make sure that you substitute a value for `principal` that matches your particular environment.

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  useTicketCache=false
  keyTab="/etc/solr/conf/solr.keytab"
  principal="solr/fully.qualified.domain.name@<YOUR-REALM>";
};
```

Enable ZooKeeper ACLs

To enable Solr to work with a secured ZooKeeper quorum, you must enable ACLs on the `/solr` znode:

1. Shut down the Solr cluster.
2.

Enabling LDAP Authentication for Solr

Before continuing, make sure that you have completed the steps in Enabling Kerberos Authentication for Solr on page 152. Solr supports LDAP authentication for external Solr client including:

- Command-line tools
- curl
- Web browsers

- Solr Java clients

In some cases, Solr does not support LDAP authentication. Use Kerberos authentication instead in these cases. Solr does not support LDAP authentication with:

- Search indexing components including the MapReduce indexer, Lily HBase indexer, or Flume.
- Solr internal requests such as those for replication or querying.
- Hadoop delegation token management requests such as `GETDELEGATIONTOKEN` or `RENEWDELEGATIONTOKEN`.

Configuring LDAP Authentication for Solr using Cloudera Manager

You can configure LDAP-based authentication using Cloudera Manager at the Solr service level.

1. Go to the **Solr** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Solr**
4. Select **Category** > **Security**
5. Select **Enable LDAP**.
6. Enter the LDAP URI in the **LDAP URI** property.
7. Configure only one of following mutually exclusive parameters:

    - **LDAP BaseDN**: Replaces the username with a "distinguished name" (DN) of the form: `uid=userid,ldap_baseDN`. Typically used for OpenLDAP server installation.

    -OR-

    - **LDAP Domain**: Replaces the username with a string `username@ldap_domain`. Typically used for Active Directory server installation.

Configuring LDAP Authentication for Solr Using the Command Line

To enable LDAP authentication using the command line, configure the following environment variables in `/etc/default/solr`:

```
SOLR_AUTHENTICATION_HTTP_SCHEMES=Negotiate,Basic
SOLR_AUTHENTICATION_HTTP_DELEGATION_MGMT_SCHEMES=Negotiate
SOLR_AUTHENTICATION_HTTP_BASIC_HANDLER=ldap
SOLR_AUTHENTICATION_HTTP_NEGOTIATE_HANDLER=kerberos
SOLR_AUTHENTICATION_LDAP_PROVIDER_URL=ldap://www.example.com

# Specify value for only one of SOLR_AUTHENTICATION_LDAP_BASE_DN or
SOLR_AUTHENTICATION_LDAP_BIND_DOMAIN property.
SOLR_AUTHENTICATION_LDAP_BASE_DN=ou=Users,dc=example,dc=com
# SOLR_AUTHENTICATION_LDAP_BIND_DOMAIN=
# Required when using 'Start TLS' extension
# SOLR_AUTHENTICATION_LDAP_ENABLE_START_TLS=false
```

Securing LDAP Connections

You can secure communications using LDAP-based encryption.

To avoid sending credentials over the wire in clear-text, you must configure a secure connection between both the client and Solr, and between Solr and the LDAP server. The secure connection could use SSL or TLS.

Secure LDAP connections through SSL:

For SSL-enabled LDAP connections, specify a prefix of `ldaps://` instead of `ldap://`. Also, the default port for SSL-enabled LDAP connections is 636 instead of 389.

Secure LDAP connections through TLS:

TLS, the successor to the SSL protocol, is supported by most modern LDAP servers. Unlike SSL connections, TLS connections can be made on the same server port as non-TLS connections. You can enable xxx using Cloudera Manager.

1. Go to the **Solr** service.
2. Click the **Configuration** tab.

3. Select **Scope** > **Solr**
4. Select **Category** > **Security**
5. Select **Enable LDAP TLS**.
6. Import the LDAP server security certificate in the Solr Trust Store file:
    a. Enter the location for the Solr Trust Store File in **Solr TLS/SSL Certificate Trust Store File**.
    b. Enter the password for the Solr Trust Store File in **Solr TLS/SSL Certificate Trust Store Password**.

LDAP Client Configuration

Some HTTP clients such as curl or the Apache Http Java client must be configured to use a particular scheme. For example:

- curl tool supports using Kerberos or username/password authentication. Kerberos is activated using the `--negotiate` flag and username/password based authentication is activated using the `--basic` and `-u` flags.
- Apache HttpClient library can be configured to use specific authentication scheme. For more information, see the HTTP authentication chapter of Apache's HttpClient Tutorial.

Typically, web browsers automatically choose a preferred authentication scheme. For more information, see the HTTP authentication topic in The Chromium Projects.

To use LDAP authentication with Solr Java clients, `HttpClientConfigurer` needs to configured for Solr. This can either be done programmatically or using Java system properties.

For example, programmatic initialization might appear as:

SampleSolrClient.java

```
import org.apache.solr.client.solrj.impl.HttpClientUtil;
import org.apache.solr.client.solrj.impl.PreemptiveBasicAuthConfigurer;
import org.apache.solr.common.params.ModifiableSolrParams;

/**
 * This method initializes the Solr client to use LDAP authentication
 * This configuration is applicable to all Solr clients.
 * @param ldapUserName LDAP user name
 * @param ldapPassword  LDAP user password
 */
public static void initialize(String ldapUserName, String ldapPassword) {
  HttpClientUtil.setConfigurer(new PreemptiveBasicAuthConfigurer());
  ModifiableSolrParams params = new ModifiableSolrParams();
  params.set(HttpClientUtil.PROP_BASIC_AUTH_USER, ldapUserName);
  params.set(HttpClientUtil.PROP_BASIC_AUTH_PASS, ldapPassword);
  // Configure the JVM default parameters.
  PreemptiveBasicAuthConfigurer.setDefaultSolrParams(params);
}
```

For configuration using system properties, configure the following system properties:

Table 6: System properties configuration for LDAP authentication

System property	Description
`solr.httpclient.configurer`	Fully qualified classname of `HttpClientConfigurer` implementation. For example, `org.apache.solr.client.solrj.impl.PreemptiveBasicAuthConfigurer`.
`solr.httpclient.config`	Http client configuration properties file path. For example, `ldap-credentials.properties`.

For example, the entry in `ldap-credentials.properties` might appear as:

ldap-credentials.properties

```
httpBasicAuthUser=user1
httpBasicAuthPassword=passwd
```

Using Kerberos with Solr

The process of enabling Solr clients to authenticate with a secure Solr is specific to the client. This section demonstrates:

- Using Kerberos and curl
- Using solrctl
- Configuring SolrJ Library Usage
- This enables technologies including:
  - Command line solutions
  - Java applications
  - The MapReduceIndexerTool
- Configuring Flume Morphline Solr Sink Usage

Secure Solr requires that the CDH components that it interacts with are also secure. Secure Solr interacts with HDFS, ZooKeeper and optionally HBase, MapReduce, and Flume.

Using Kerberos and curl

You can use Kerberos authentication with clients such as `curl`. To use `curl`, begin by acquiring valid Kerberos credentials and then run the desired command. For example, you might use commands similar to the following:

```
$ kinit -kt username.keytab username
$ curl --negotiate -u foo:bar http://solrserver:8983/solr/
```

> **Note:** Depending on the tool used to connect, additional arguments may be required. For example, with `curl`, `--negotiate` and `-u` are required. The username and password specified with `-u` is not actually checked because Kerberos is used. As a result, any value such as foo:bar or even just : is acceptable. While any value can be provided for `-u`, note that the option is required. Omitting `-u` results in a 401 Unauthorized error, even though the `-u` value is not actually used.

Using solrctl

If you are using `solrctl` to manage your deployment in an environment that requires Kerberos authentication, you must have valid Kerberos credentials, which you can get using `kinit`. For more information on `solrctl`, see solrctl Reference

Configuring SolrJ Library Usage

If using applications that use the solrj library, begin by establishing a Java Authentication and Authorization Service (JAAS) configuration file.

Create a JAAS file:

- If you have already used `kinit` to get credentials, you can have the client use those credentials. In such a case, modify your `jaas-client.conf` file to appear as follows:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=false
 useTicketCache=true
 principal="user/fully.qualified.domain.name@<YOUR-REALM>";
 };
```

where `user/fully.qualified.domain.name@<YOUR-REALM>` is replaced with your credentials.

- If you want the client application to authenticate using a keytab, modify `jaas-client.conf` as follows:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="/path/to/user.keytab"
 storeKey=true
 useTicketCache=false
 principal="user/fully.qualified.domain.name@EXAMPLE.COM";
};
```

  Replace `/path/to/user.keytab` with the keytab file you want to use and `user/fully.qualified.domain.name@EXAMPLE.COM` with the principal in the keytab. If the principal omits the hostname, omit it in the `jaas-client.conf` file as well (for example, `jdoe@EXAMPLE.COM`).

Use the JAAS file to enable solutions:

- **Command line solutions**

  Set the property when invoking the program. For example, if you were using a jar, you might use:

```
java -Djava.security.auth.login.config=/home/user/jaas-client.conf -jar app.jar
```

- **Java applications**

  Set the Java system property `java.security.auth.login.config`. For example, if the JAAS configuration file is located on the filesystem as `/home/user/jaas-client.conf`. The Java system property `java.security.auth.login.config` must be set to point to this file. Setting a Java system property can be done programmatically, for example using a call such as:

```
System.setProperty("java.security.auth.login.config", "/home/user/jaas-client.conf");
```

- **The MapReduceIndexerTool**

  The MapReduceIndexerTool uses SolrJ to pass the JAAS configuration file. Using the MapReduceIndexerTool in a secure environment requires the use of the `HADOOP_OPTS` variable to specify the JAAS configuration file. For example, you might issue a command such as the following:

```
HADOOP_OPTS="-Djava.security.auth.login.config=/home/user/jaas.conf" \
hadoop jar MapReduceIndexerTool
```

- **Configuring the hbase-indexer CLI**

  Certain hbase-indexer CLI commands such as `replication-status` attempt to read ZooKeeper hosts owned by HBase. To successfully use these commands in Solr in a secure environment, specify a JAAS configuration file with the HBase principal in the `HBASE_INDEXER_OPTS` environment variable. For example, you might issue a command such as the following:

```
HBASE_INDEXER_OPTS="-Djava.security.auth.login.config=/home/user/hbase-jaas.conf" \
hbase-indexer replication-status
```

Configuring Flume Morphline Solr Sink Usage

Repeat this process on all Flume hosts:

1. If you have not created a keytab file, do so now at `/etc/flume-ng/conf/flume.keytab`. This file should contain the service principal `flume/<fully.qualified.domain.name>@<YOUR-REALM>`. See Flume Authentication on page 112 for more information.

2. Create a JAAS configuration file for flume at `/etc/flume-ng/conf/jaas-client.conf`. The file should appear as follows:

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 useTicketCache=false
 keyTab="/etc/flume-ng/conf/flume.keytab"
 principal="flume/<fully.qualified.domain.name>@<YOUR-REALM>";
};
```

3. Add the flume JAAS configuration to the `JAVA_OPTS` in `/etc/flume-ng/conf/flume-env.sh`. For example, you might change:

```
JAVA_OPTS="-Xmx500m"
```

to:

```
JAVA_OPTS="-Xmx500m -Djava.security.auth.login.config=/etc/flume-ng/conf/jaas-client.conf"
```

Spark Authentication

Spark has an internal mechanism that authenticates executors with the driver controlling a given application. This mechanism is enabled using the Cloudera Manager Admin Console, as detailed in Enabling Spark Authentication.

When Spark on YARN is running on a secure cluster, users must authenticate to Kerberos before submitting jobs, as detailed in Running Spark Applications on Secure Clusters.

Enabling Spark Authentication

Minimum Required Role: Security Administrator (also provided by **Full Administrator**)

Spark has an internal mechanism that authenticates executors with the driver controlling a given application. This mechanism is enabled using the Cloudera Manager Admin Console, as detailed below. Cluster administrators can enable the `spark.authenticate` mechanism to authenticate the various processes that support a Spark application.

To enable this feature on the cluster:

1. Log into the Cloudera Manager Admin Console.
2. Select **Clusters** > **Spark** (or **Clusters** > **Spark_on_YARN**).
3. Click the **Configuration** menu.
4. Scroll down to the **Spark Authentication** setting, or search for `spark.authenticate` to find it.
5. In the Spark Authentication setting, click the checkbox next to the **Spark (Service-Wide)** property to activate the setting.
6. Click **Save Changes**.

> **Note:** If your cluster supports Spark 2, you must make the same change to the Spark 2 setting. For example:

7. Restart YARN:
   - Select **Clusters** > **YARN**.
   - Select **Restart** from the **Actions** drop-down selector.
8. Re-deploy the client configurations:
   - Select **Clusters** > ***Cluster_name***
   - Select **Deploy Client Configurations** from the **Actions** drop-down selector.

Running Spark Applications on Secure Clusters

Secure clusters are clusters that use Kerberos for authentication. For secure clusters, Spark History Server automatically uses Kerberos, so there's nothing to configure.

Users running Spark applications must first authenticate to Kerberos, using `kinit`, as follows:

```
$ kinit
ldap@EXAMPLE-REALM.COM:'s password:
```

After authenticating to Kerberos, users can submit their applications using `spark-submit` as usual, as shown below. This command submits one of the default Spark sample jobs using an environment variable as part of the path, so modify as needed for your own use:

```
$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn \
--deploy-mode cluster $SPARK_HOME/lib/spark-examples.jar 10
```

Configuring Spark on YARN for Long-Running Applications

Long-running applications such as Spark Streaming jobs must be able to write to HDFS, which means that the `hdfs` user may need to delegate tokens possibly beyond the default lifetime. This workload type requires passing Kerberos principal and keytab to the `spark-submit` script using the `--principal` and `--keytab` parameters. The keytab is copied to the host running the ApplicationMaster, and the Kerberos login is renewed periodically by using the principal and keytab to generate the required delegation tokens needed for HDFS.

> **Note:** For secure distribution of the keytab to the ApplicationMaster host, the cluster should be configured for TLS/SSL communication for YARN and HDFS encryption .

Create the Spark Principal and Keytab File

These are needed for long-running applications running on Spark on YARN `cluster` mode only.

1. Create the `spark` principal and `spark.keytab` file:

```
kadmin: addprinc -randkey spark/fully.qualified.domain.name@YOUR-REALM.COM
kadmin: xst -k spark.keytab spark/fully.qualified.domain.name
```

See Step 4: Create and Deploy the Kerberos Principals and Keytab Files on page 91 for more information about Kerberos and its use with Cloudera clusters.

Sqoop 2 Authentication

This section describes how to configure Sqoop 2 with Kerberos security in a Hadoop cluster.

> **Note:** Sqoop 2 is being deprecated. Cloudera recommends using Sqoop 1.

Create the Sqoop 2 Principal and Keytab File

You need to create a `sqoop2.keytab` file for Sqoop 2. Follow these steps:

1. Create the principal and keytab file:

```
kadmin: addprinc -randkey sqoop2/fully.qualified.domain.name@YOUR-REALM.COM
kadmin: xst -k sqoop2.keytab sqoop2/fully.qualified.domain.name
```

2. Move the file into the Sqoop 2 configuration directory and restrict its access exclusively to the sqoop2 user:

```
$ mv sqoop2.keytab /etc/sqoop2/conf/
$ chown sqoop2 /etc/sqoop2/conf/sqoop2.keytab
$ chmod 400 /etc/sqoop2/conf/sqoop2.keytab
```

For more details on creating Kerberos principals and keytabs, see Step 4: Create and Deploy the Kerberos Principals and Keytab Files on page 91.

Configure Sqoop 2 to Use Kerberos

Edit the Sqoop 2 configuration file `sqoop.properties` file in the `/etc/sqoop2/conf` directory and add the following properties:

```
org.apache.sqoop.authentication.type=KERBEROS
org.apache.sqoop.authentication.handler=org.apache.sqoop.security.KerberosAuthenticationHandler
org.apache.sqoop.authentication.kerberos.principal=sqoop2/fully.qualified.domain.name@YOUR-REALM.COM
org.apache.sqoop.authentication.kerberos.keytab=/etc/sqoop2/conf/sqoop2.keytab
```

Sqoop 1, Pig, and Whirr Security

Here is a summary of some security features for other CDH 5 components.

Pig

Supports security with no configuration required.

Sqoop 1

Sqoop1 does not support:

- TLS/SSL connections to Oracle, MySQL, or other databases
- Kerberos authentication to external databases

Whirr

Whirr does not support security in CDH 5.

ZooKeeper Authentication

As of Cloudera Manager 5.11, ZooKeeper supports mutual server-to-server (quorum peer) authentication using SASL (Simple Authentication and Security Layer), which provides a layer around Kerberos authentication. Server to server authentication among ZooKeeper servers in an ensemble mitigates the risk of spoofing by a rogue server on an unsecured network. For more information about quorum peer authentication and how the feature leverages ZooKeeper's SASL support, see the Cloudera Engineering Blog post, Hardening Apache ZooKeeper Security.

Client-to-server SASL-based authentication has been supported since Cloudera Manager/CDH 5.2 (ZooKeeper 3.4.0+). Follow the steps in Configuring ZooKeeper Server for Kerberos Authentication on page 161 and Configuring ZooKeeper Client Shell for Kerberos Authentication on page 162 to configure ZooKeeper to use this mechanism.

To configure client-server or server-server authentication for ZooKeeper, follow the appropriate steps below:

Requirements

Configuring ZooKeeper to use Kerberos for client-server or server-server authentication requires that your organization's Kerberos instance (MIT Kerberos, Microsoft Active Directory) be up and running, and reachable by the ZooKeeper server or client during the configuration processes detailed below. See Configuring Hadoop Security in CDH 5 for details.

Before enabling mutual authentication, the ZooKeeper servers in the cluster must be configured to authenticate using Kerberos.

Note: Cloudera recommends that you ensure your ZooKeeper ensemble is working properly, before you attempt to integrate Kerberos authentication. See ZooKeeper Installation for details.

Configuring ZooKeeper Server for Kerberos Authentication

Follow the steps below for each ZooKeeper server in the ensemble. To maintain consistency across ZooKeeper servers in the ensemble, use the same name for the keytab file you deploy to each server, for example, `zookeeper.keytab` (step 2, below). Each keytab file will contain its respective host's fully-qualified domain name (FQDN).

1. Create a service principal for the ZooKeeper server using the fully-qualified domain name (FQDN) of the host on which ZooKeeper server is running and the name of your Kerberos realm using the pattern `zookeeper/`*`fqdn.example.com@ YOUR-REALM`*. This principal will be used to authenticate the ZooKeeper server with the Hadoop cluster. Create this service principal as follows:

```
kadmin: addprinc -randkey zookeeper/fqdn.example.com@YOUR-REALM
```

2. Create a keytab file for the ZooKeeper server:

```
$ kadmin
kadmin: xst -k zookeeper.keytab zookeeper/fqdn.example.com@YOUR-REALM
```

Note: For consistency across ZooKeeper Servers, use the same name for the keytab file you create for each subsequent ZooKeeper Server host system you configure using these steps, for example, `zookeeper.keytab`.

3. Copy the `zookeeper.keytab` file to the ZooKeeper configuration directory on the ZooKeeper server host, using the appropriate ZooKeeper configuration directory for your installation type, as follows:

   - Package installation: `/etc/zookeeper/conf/`
   - Tarball installation: *`EXPANDED_DIR`*`/conf`

   The `zookeeper.keytab` file should be owned by the `zookeeper` user, with owner-only read permissions.
4. Add the following lines to the ZooKeeper configuration file `zoo.cfg`:

```
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
jaasLoginRenew=3600000
```

5. Set up the Java Authentication and Authorization Service (JAAS) by creating a `jaas.conf` file in the ZooKeeper configuration directory with the settings shown below, replacing *`fqdn.example.com`* with the ZooKeeper server's hostname.

```
Server {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="/etc/zookeeper/conf/zookeeper.keytab"
  storeKey=true
  useTicketCache=false
  principal="zookeeper/fqdn.example.com
  @YOUR-REALM";
};
```

6. Add the following setting to the `java.env` file located in the ZooKeeper configuration directory, creating the file if necessary:

```
export JVMFLAGS="-Djava.security.auth.login.config=/etc/zookeeper/conf/jaas.conf"
```

7. Repeat these steps for each ZooKeeper server in the ensemble.
8. Restart the ZooKeeper server to have the configuration changes take effect. See ZooKeeper Installation for details.

Configuring ZooKeeper Client Shell for Kerberos Authentication

In addition to configuring ZooKeeper Server hosts to use Kerberos for authentication, you should also configure the ZooKeeper client shell (the ZooKeeper CLI) to authenticate to the ZooKeeper service using Kerberos credentials. As with the ZooKeeper Server, you must create a Kerberos principal for the client, as detailed below:

1. Create a Kerberos principal for the `zookeeper-client`, `zkcli@YOUR-REALM`, replacing *YOUR-REALM* with the name of your organization's Kerberos realm:

```
kadmin: addprinc -randkey zkcli@YOUR-REALM
```

2. Create a keytab file for the ZooKeeper client shell using the `-norandkey` option.

> **Note:** Not all versions of `kadmin` support the `-norandkey` option, in which case, simply omit this option from the command. Using the `kadmin` command without the `-norandkey` option invalidates previously exported keytabs and generates a new password.

```
$ kadmin
kadmin: xst -norandkey -k zkcli.keytab zkcli@YOUR-REALM
```

3. On the host running the ZooKeeper client shell, set up JAAS (Java Authentication and Authorization Service) in the configuration directory appropriate for your installation type:
   - Package installation: `/etc/zookeeper/conf/`
   - Tarball installation: `EXPANDED_DIR/conf`
4. Create a `jaas.conf` file containing the following settings:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="/path/to/zkcli.keytab"
  storeKey=true
  useTicketCache=false
  principal="zkcli@YOUR-REALM";
};
```

5. In this same configuration directory, add the following setting to the `java.env` file, creating the file if necessary.

```
export JVMFLAGS="-Djava.security.auth.login.config=/etc/zookeeper/conf/jaas.conf"
```

Verifying the Configuration

After enabling Kerberos authentication and restarting the ZooKeeper cluster, you can verify that the authentication is working correctly by following these steps:

1. Start the ZooKeeper client, passing to it the name of a ZooKeeper server:

```
zookeeper-client -server fqdn.example.com:port
```

2. From the ZooKeeper CLI, create a protected znode using your ZooKeeper client principal:

```
create /znode1 znode1data sasl:zkcli@{{YOUR-REALM}}:cdwra
```

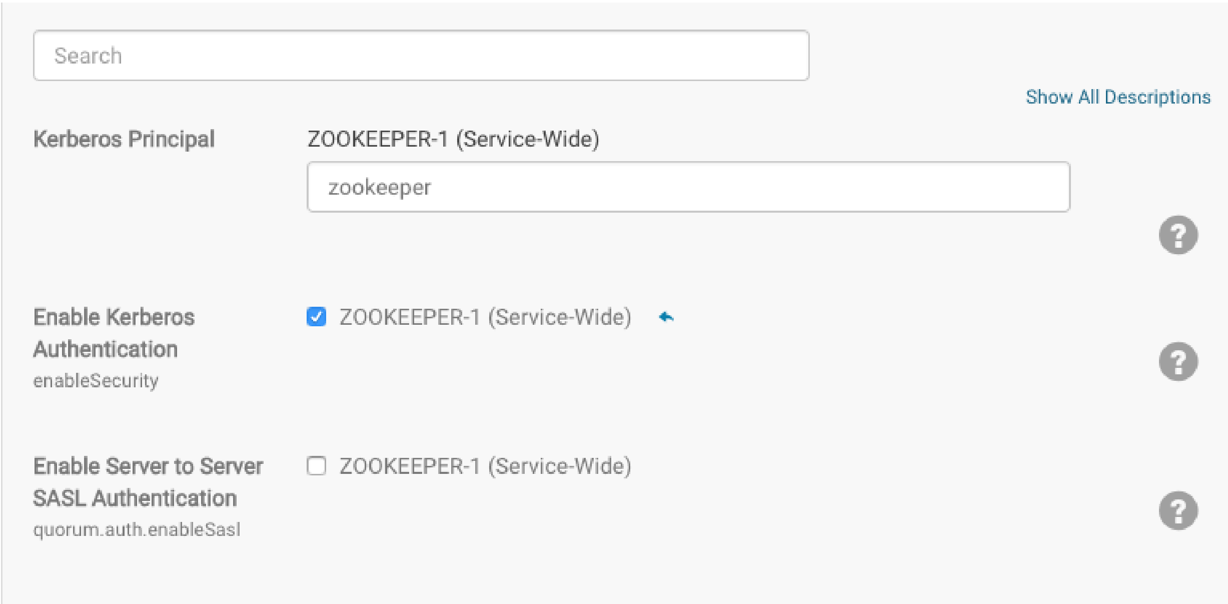
3. Verify the znode is created and the ACL is set correctly:

```
getAcl /znode1
```

The `getAcl` command returns the znode's scheme and permissions values. Verify that these are as expected.

Enabling Server-Server Mutual Authentication

As of Cloudera Manager 5.11, support for mutual authentication between ZooKeeper Servers can be enabled through the Cloudera Manager Admin Console. For secured networks, server-to-server authentication is considered an optional security enhancement, so the capability is disabled by default:



Server-to-server SASL authentication requires all servers in the ZooKeeper ensemble to authenticate using Kerberos, as detailed in Configuring ZooKeeper Server for Kerberos Authentication on page 161.

Assuming your cluster is already configured to authenticate using Kerberos, you can enable mutual authentication as follows:

1. Log into the Cloudera Manager Admin Console.
2. Select **Clusters** > **ZOOKEEPER-n**.
3. Click the **Configuration** tab.
4. Select **Category** > **Security** under the Filters menu to display the security properties.
5. Click the **Enable Server to Server SASL Authentication** box to select it.
6. Click **Save**.
7. Select **Restart** from the **Actions** drop-down to restart the cluster with this setting.

To disable server-to-server SASL authentication, simply return to the Cloudera Manager Admin Console page shown above, de-select **Enable Server to Server SASL Authentication** (by clicking the checked box), and restart the cluster.

Hadoop Users in Cloudera Manager and CDH

During the Cloudera Manager/CDH installation process, several Linux user accounts and groups (listed in the table below) are created by default. If you configure the cluster to integrate with Kerberos for authentication you must create Kerberos principals and keytab files for the user accounts, as noted in the table.

> **Note:** Cloudera Manager 5.3 (and later releases) can be setup for ***single user mode***. In single user mode, Hadoop users and groups are subsumed by `cloudera-scm:cloudera-scm`. Cloudera Manager starts all Cloudera Manager Agent processes and services running on the nodes in the cluster as a unit, owned by this `cloudera-scm:cloudera-scm`. Single user mode is not recommended for production clusters.

Table 7: Users and Groups

Component (Version)	Unix User ID	Groups	Notes
Cloudera Manager (all versions)	`cloudera-scm`	`cloudera-scm`	Cloudera Manager processes such as the Cloudera Manager Server and the monitoring roles run as this user. The Cloudera Manager keytab file must be named `cmf.keytab` since that name is hard-coded in Cloudera Manager. **Note:** Applicable to clusters managed by Cloudera Manager only.
Apache Accumulo (Accumulo 1.4.3 and higher)	`accumulo`	`accumulo`	Accumulo processes run as this user.
Apache Avro			No special users.
Apache Flume	`flume`	`flume`	The sink that writes to HDFS as this user must have write privileges.
Apache HBase	`hbase`	`hbase`	The Master and the RegionServer processes run as this user.
HDFS	`hdfs`	`hdfs, hadoop`	The NameNode and DataNodes run as this user, and the HDFS root directory as well as the directories used for edit logs should be owned by it.
Apache Hive	`hive`	`hive`	The HiveServer2 process and the Hive Metastore processes run as this user. A user must be defined for Hive access to its Metastore DB (for example, MySQL or Postgres) but it can be any identifier and does not correspond to a Unix uid. This is `javax.jdo.option.ConnectionUserName` in hive-site.xml.
Apache HCatalog	`hive`	`hive`	The WebHCat service (for REST access to Hive functionality) runs as the `hive` user.
HttpFS	`httpfs`	`httpfs`	The HttpFS service runs as this user. See HttpFS Security Configuration for instructions on how to generate the merged `httpfs-http.keytab` file.
Hue	`hue`	`hue`	Hue services run as this user.
Hue Load Balancer (Cloudera Manager 5.5 and higher)	apache	apache	The Hue Load balancer has a dependency on the apache2 package that uses the apache user name. Cloudera Manager does not run processes using this user ID.
Cloudera Impala	`impala`	`impala, hive`	Impala services run as this user.
Apache Kafka (Cloudera	`kafka`	`kafka`	Kafka services run as this user.

Component (Version)	Unix User ID	Groups	Notes
Distribution of Kafka 1.2.0)			
Java KeyStore KMS (CDH 5.2.1 and higher)	`kms`	`kms`	The Java KeyStore KMS service runs as this user.
Key Trustee KMS (CDH 5.3 and higher)	`kms`	`kms`	The Key Trustee KMS service runs as this user.
Key Trustee Server (CDH 5.4 and higher)	`keytrustee`	`keytrustee`	The Key Trustee Server service runs as this user.
Kudu	`kudu`	`kudu`	Kudu services run as this user.
Llama	`llama`	`llama`	Llama runs as this user.
Apache Mahout			No special users.
MapReduce	`mapred`	`mapred, hadoop`	Without Kerberos, the JobTracker and tasks run as this user. The LinuxTaskController binary is owned by this user for Kerberos.
Apache Oozie	`oozie`	`oozie`	The Oozie service runs as this user.
Parquet			No special users.
Apache Pig			No special users.
Cloudera Search	`solr`	`solr`	The Solr processes run as this user.
Apache Spark	`spark`	`spark`	The Spark History Server process runs as this user.
Apache Sentry	`sentry`	`sentry`	The Sentry service runs as this user.
Apache Sqoop	`sqoop`	`sqoop`	This user is only for the Sqoop1 Metastore, a configuration option that is not recommended.
Apache Sqoop2	`sqoop2`	`sqoop, sqoop2`	The Sqoop2 service runs as this user.
Apache Whirr			No special users.
YARN	`yarn`	`yarn, hadoop`	Without Kerberos, all YARN services and applications run as this user. The LinuxContainerExecutor binary is owned by this user for Kerberos.
Apache ZooKeeper	`zookeeper`	`zookeeper`	The ZooKeeper processes run as this user. It is not configurable.

Keytabs and Keytab File Permissions

Linux `user` accounts, such as `hdfs`, `flume`, or `mapred` are mapped to the `username` portion of the Kerberos principal names, as follows:

```
username/fqdn.example.com@YOUR-REALM.COM
```

For example, the Kerberos principal for Apache Flume would be:

```
flume/fqdn.example.com@YOUR-REALM.COM
```

Keytabs that contain multiple principals are merged automatically from individual keytabs by Cloudera Manager. If you do not use Cloudera Manager, you must merge the keytabs manually.

The table below lists the usernames to use for Kerberos principal names.

Table 8: Clusters Managed by Cloudera Manager

Component (Unix User ID)	Service	Kerberos Principals	Filename (*.keytab)	Keytab File Owner	Keytab File Group	File Permission (octal)
Cloudera Manager (`cloudera-scm`)	NA	cloudera-scm	cmf	cloudera-scm	cloudera-scm	600
Cloudera Management Service (`cloudera-scm`)	cloudera-mgmt-REPORTSMANAGER	hdfs	headlamp	cloudera-scm	cloudera-scm	600
Cloudera Management Service (`cloudera-scm`)	cloudera-mgmt-SERVICEMONITOR, cloudera-mgmt-ACTIVITYMONITOR	hue	cmon	cloudera-scm	cloudera-scm	600
Cloudera Management Service (`cloudera-scm`)	cloudera-mgmt-HOSTMONITOR	N/A	N/A	N/A	N/A	N/A
Apache Accumulo (`accumulo`)	accumulo16-ACCUMULO16_MASTER	accumulo	accumulo16	cloudera-scm	cloudera-scm	600
	accumulo16-ACCUMULO16_TRACER					
	accumulo16-ACCUMULO16_MONITOR					
	accumulo16-ACCUMULO16_GC					
	accumulo16-ACCUMULO16_TSERVER					
Flume (`flume`)	flume-AGENT	flume	flume	cloudera-scm	cloudera-scm	600
HBase (`hbase`)	hbase-HBASETHRIFTSERVER	HTTP	HTTP	cloudera-scm	cloudera-scm	600
	hbase-REGIONSERVER	hbase	hbase			
	hbase-HBASERESTSERVER					
	hbase-MASTER					
HDFS (`hdfs`)	hdfs-NAMENODE	hdfs, HTTP	hdfs	cloudera-scm	cloudera-scm	600
	hdfs-DATANODE					
	hdfs-SECONDARYNAMENODE					
Hive (`hive`)	hive-HIVESERVER2	hive	hive	cloudera-scm	cloudera-scm	600
	hive-WEBHCAT	HTTP	HTTP			
	hive-HIVEMETASTORE	hive	hive			
HttpFS (`httpfs`)	hdfs-HTTPFS	httpfs	httpfs	cloudera-scm	cloudera-scm	600
Hue (`hue`)	hue-KT_RENEWER	hue	hue	cloudera-scm	cloudera-scm	600
Impala (`impala`)	impala-STATESTORE	impala	impala	cloudera-scm	cloudera-scm	600

Component (Unix User ID)	Service	Kerberos Principals	Filename (*.keytab)	Keytab File Owner	Keytab File Group	File Permission (octal)
	impala-CATALOGSERVER					
	impala-IMPALAD					
Java KeyStore KMS (`kms`)	kms-KMS	HTTP	kms	cloudera-scm	cloudera-scm	600
Apache Kafka (`kafka`)	kafka-KAFKA_BROKER	kafka	kafka	kafka	kafka	600
Key Trustee KMS (`kms`)	keytrustee-KMS_KEYTRUSTEE	HTTP	keytrustee	cloudera-scm	cloudera-scm	600
Llama (`llama`)	impala-LLAMA	llama, HTTP	llama	cloudera-scm	cloudera-scm	600
MapReduce (`mapred`)	mapreduce-JOBTRACKER	mapred, HTTP	mapred	cloudera-scm	cloudera-scm	600
	mapreduce- TASKTRACKER					
Oozie (`oozie`)	oozie-OOZIE_SERVER	oozie, HTTP	oozie	cloudera-scm	cloudera-scm	600
Search (`solr`)	solr-SOLR_SERVER	solr, HTTP	solr	cloudera-scm	cloudera-scm	600
Sentry (`sentry`)	sentry-SENTRY_SERVER	sentry	sentry	cloudera-scm	cloudera-scm	600
Spark (`spark`)	spark_on_yarn-SPARK_YARN_HISTORY_SERVER	spark	spark	cloudera-scm	cloudera-scm	600
YARN (`yarn`)	yarn-NODEMANAGER	yarn, HTTP	yarn	cloudera-scm	cloudera-scm	644
	yarn- RESOURCEMANAGER					600
	yarn-JOBHISTORY					600
ZooKeeper (`zookeeper`)	zookeeper-server	zookeeper	zookeeper	cloudera-scm	cloudera-scm	600

Table 9: CDH Clusters Not Managed by Cloudera Manager

Component (Unix User ID)	Service	Kerberos Principals	Filename (*.keytab)	Keytab File Owner	Keytab File Group	File Permission (octal)
Apache Accumulo (`accumulo`)	accumulo16-ACCUMULO16_MASTER	accumulo	accumulo16	accumulo	accumulo	600
	accumulo16-ACCUMULO16_TRACER					
	accumulo16-ACCUMULO16_MONITOR					
	accumulo16-ACCUMULO16_GC					
	accumulo16-ACCUMULO16_TSERVER					
Flume (`flume`)	flume-AGENT	flume	flume	flume	flume	600
HBase (`hbase`)	hbase-HBASETHRIFTSERVER	HTTP	HTTP	hbase	hbase	600
	hbase-REGIONSERVER	hbase	hbase			
	hbase-HBASERESTSERVER					
	hbase-MASTER					
HDFS (`hdfs`)	hdfs-NAMENODE	hdfs, HTTP	hdfs	hdfs	hdfs	600

Component (Unix User ID)	Service	Kerberos Principals	Filename (*.keytab)	Keytab File Owner	Keytab File Group	File Permission (octal)
	hdfs-DATANODE					
	hdfs- SECONDARYNAMENODE					
Hive (hive)	hive-HIVESERVER2	hive	hive	hive	hive	600
	hive-WEBHCAT	HTTP	HTTP			
	hive-HIVEMETASTORE	hive	hive			
HttpFS (httpfs)	hdfs-HTTPFS	httpfs	httpfs	httpfs	httpfs	600
Hue (hue)	hue-KT_RENEWER	hue	hue	hue	hue	600
Impala (impala)	impala-STATESTORE	impala	impala	impala	impala	600
	impala-CATALOGSERVER					
	impala-IMPALAD					
Llama (llama)	impala-LLAMA	llama, HTTP	llama	llama	llama	600
Java KeyStore KMS (kms)	kms-KMS	HTTP	kms	kms	kms	600
Apache Kafka (kafka)	kafka-KAFKA_BROKER	kafka	kafka	kafka	kafka	600
Key Trustee KMS (kms)	kms-KEYTRUSTEE	HTTP	kms	kms	kms	600
MapReduce (mapred)	mapreduce-JOBTRACKER	mapred, HTTP	mapred	mapred	hadoop	600
	mapreduce- TASKTRACKER					
Oozie (oozie)	oozie-OOZIE_SERVER	oozie, HTTP	oozie	oozie	oozie	600
Search (solr)	solr-SOLR_SERVER	solr, HTTP	solr	solr	solr	600
Sentry (sentry)	sentry-SENTRY_SERVER	sentry	sentry	sentry	sentry	600
Spark (spark)	spark_on_yarn-SPARK_YARN_HISTORY_SERVER	spark	spark	spark	spark	600
YARN (yarn)	yarn-NODEMANAGER	yarn, HTTP	yarn	yarn	hadoop	644
	yarn- RESOURCEMANAGER					600
	yarn-JOBHISTORY					600
ZooKeeper (zookeeper)	zookeeper-server	zookeeper	zookeeper	zookeeper	zookeeper	600

Configuring a Cluster-dedicated MIT KDC with Cross-Realm Trust

If you use Cloudera Manager to enable Hadoop security on your cluster, the Cloudera Manager Server will create several principals and then generate keytabs for those principals. Cloudera Manager will then deploy the keytab files on every host in the cluster. See Hadoop Users in Cloudera Manager and CDH on page 163 for a complete listing of the principals created by Cloudera Manager.

> **Note:** The following instructions illustrate an example of creating and deploying the principals and keytab files for MIT Kerberos. (If you are using another version of Kerberos, refer to the Kerberos documentation for the version of the operating system you are using, for instructions.)

When to use kadmin.local and kadmin

When performing the Kerberos commands in this document, you can use `kadmin.local` or `kadmin` depending on your access and account:

- If you can log on to the KDC host directly, and have root access or a Kerberos admin account, use the `kadmin.local` command.
- When accessing the KDC from a remote host, use the `kadmin` command.

To start `kadmin.local` on the KDC host:

```
$ sudo kadmin.local
```

To run `kadmin` from any host:

```
$ kadmin
```

> **Note:**
>
> - In this guide, `kadmin` is shown as the prompt for commands in the `kadmin` shell, but you can type the same commands at the `kadmin.local` prompt in the `kadmin.local` shell.
> - Running `kadmin.local` may prompt you for a password because it is being run using `sudo`. You should provide your Unix password. Running `kadmin` may prompt you for a password because you need Kerberos admin privileges. You should provide your Kerberos admin password.

Setting up a Cluster-Dedicated KDC and Default Realm for the Hadoop Cluster

Cloudera has tested the following configuration approaches to Kerberos security for clusters managed by Cloudera Manager. For administration teams that are just getting started with Kerberos security, we recommend starting with these approaches to the configuration of KDC services for a number of reasons.

The number of Service Principal Names (SPNs) that are created and managed by the Cloudera Manager server for a CDH cluster can be significant, so it is important to realize the potential impact on cluster uptime and overall operations if you choose to manage keytabs manually instead. The Cloudera Manager server manages the creation of service keytabs on the proper hosts based on the current configuration of the database. Manual keytab management can be error prone and introduce delays when deploying or moving services within the cluster, especially under time-sensitive conditions.

Cloudera Manager creates SPNs within a KDC that it can access with the `kadmin` command based on configuration of the `/etc/krb5.conf` file on the Cloudera Manager host. SPNs are created with the format `service-name/host.fqdn.name@EXAMPLE.COM` where `service-name` is the relevant CDH service name such as `hue` or `hbase` or `hdfs`.

If your site already has a working KDC, and any existing principals share the same name as any of the principals that Cloudera Manager creates, the Cloudera Manager Server generates a new randomized key for those principals, and consequently causes existing keytabs to become invalid.

This is why Cloudera recommends using a dedicated local MIT Kerberos KDC and realm for the Hadoop cluster. You can set up a one-way cross-realm trust from the cluster-dedicated KDC and realm to your existing central MIT Kerberos KDC, or to an existing Active Directory realm. Using this method, there is no need to create Hadoop service principals in the central MIT Kerberos KDC or in Active Directory, but principals (users) in the central MIT KDC or in Active Directory can be authenticated to Hadoop. The steps to implement this approach are as follows:

1. Install and configure a cluster-dedicated MIT Kerberos KDC that will be managed by Cloudera Manager for creating and storing the service principals for your Hadoop cluster.

> **Note:** The `krb5-server` package includes a `logrotate` policy file to rotate log files monthly. To take advantage of this, install the `logrotate` package. No additional configuration is necessary.

2. See the example `kdc.conf` and `krb5.conf` files in Sample Kerberos Configuration Files on page 504 for configuration considerations for the KDC and Kerberos clients.
3. Configure a default Kerberos realm for the cluster you want Cloudera Manager to manage and set up one-way cross-realm trust between the cluster-dedicated KDC and either your central KDC or Active Directory. Follow the appropriate instructions below for your deployment: Using a Cluster-Dedicated KDC with a Central MIT KDC on page 170 or Using a Cluster-Dedicated MIT KDC with Active Directory on page 171.

Cloudera strongly recommends the method above because:

- It requires minimal configuration in Active Directory.
- It is comparatively easy to script the creation of many principals and keytabs. A principal and keytab must be created for every daemon in the cluster, and in a large cluster this can be extremely onerous to do directly in Active Directory.
- There is no need to involve central Active Directory administrators to get service principals created.
- It allows for incremental configuration. The Hadoop administrator can completely configure and verify the functionality the cluster independently of integrating with Active Directory.

Using a Cluster-Dedicated KDC with a Central MIT KDC

> **Important:** If you plan to use Oozie or the Hue Kerberos Ticket Renewer in your cluster, you must configure your KDC to allow tickets to be renewed, and you must configure `krb5.conf` to request renewable tickets. Typically, you can do this by adding the `max_renewable_life` setting to your realm in `kdc.conf`, and by adding the `renew_lifetime` parameter to the `libdefaults` section of `krb5.conf`. For more information about renewable tickets, see the Kerberos documentation. This is demonstrated in the Sample Kerberos Configuration Files on page 504.

1. In the `/var/kerberos/krb5kdc/kdc.conf` file on the local dedicated KDC server host, configure the default realm for the Hadoop cluster by substituting your Kerberos realm in the following `realms` property:

```
[realms]
 HADOOP.EXAMPLE.COM = {
```

2. In the `/etc/krb5.conf` file on all cluster hosts and all Hadoop client user hosts, configure the default realm for the Hadoop cluster by substituting your Kerberos realm in the following `realms` property. Also specify the local dedicated KDC server hostname in the `/etc/krb5.conf` file (for example, `kdc01.example.com`).

```
[libdefaults]
  default_realm = HADOOP.EXAMPLE.COM
[realms]
  HADOOP.EXAMPLE.COM = {
    kdc = kdc01.hadoop.example.com:88
    admin_server = kdc01.hadoop.example.com:749
    default_domain = hadoop.example.com
  }
  EXAMPLE.COM = {
    kdc = kdc01.example.com:88
    admin_server = kdc01.example.com:749
    default_domain = example.com
  }
[domain_realm]
  .hadoop.example.com = HADOOP.EXAMPLE.COM
  hadoop.example.com = HADOOP.EXAMPLE.COM
  .example.com = EXAMPLE.COM
  example.com = EXAMPLE.COM
```

3. To set up the cross-realm trust in the cluster-dedicated KDC, type the following command in the `kadmin.local` or `kadmin` shell on the cluster-dedicated KDC host to create a `krbtgt` principal. Substitute your cluster-dedicated KDC realm for `HADOOP.EXAMPLE.COM`, and substitute your central KDC realm for `EXAMPLE.COM`. Enter a trust password when prompted. Note the password because you will need to enter the exact same password in the central KDC in the next step.

```
kadmin:  addprinc krbtgt/HADOOP.EXAMPLE.COM@EXAMPLE.COM
```

4. Each of your Hadoop client users must also place this information in their local `core-site.xml` file. The easiest way to do so is by using the Cloudera Manager Admin Console to generate a client configuration file.
5. To set up the cross-realm trust in the central KDC, type the same command in the `kadmin.local` or `kadmin` shell on the central KDC host to create the exact same `krbtgt` principal and password.

```
kadmin:  addprinc krbtgt/HADOOP.EXAMPLE.COM@EXAMPLE.COM
```

> **Important:** For a cross-realm trust to operate properly, both KDCs must have the same `krbtgt` principal and password, and both KDCs must be configured to use the same encryption type.

6. To properly translate principal names from the central KDC realm into the cluster-dedicated KDC realm for the Hadoop cluster, configure the **Trusted Kerberos Realms** property of the HDFS service.
   a. Open the Cloudera Manager Admin Console.
   b. Go to the HDFS service.
   c. Click the **Configuration** tab.
   d. Select **Scope** > **HDFS (Service Wide)**
   e. Select **Category** > **Security**.
   f. Type `Kerberos` in the **Search** box.
   g. Edit the **Trusted Kerberos Realms** property to add the name of your central KDC realm. If you need to use more advanced mappings which do more than just allow principals from another domain, you may enter them in the **Additional Rules to Map Kerberos Principals to Short Names** property. For more information about name mapping rules, see Configuring the Mapping from Kerberos Principals to Short Names on page 110.
7. Each of your Hadoop client users must also place this information in their local `core-site.xml` file. The easiest way to do so is by using the Cloudera Manager Admin Console to generate a client configuration file.
8. Proceed to Step 2: If You are Using AES-256 Encryption, Install the JCE Policy File on page 69. Later in this procedure, you will restart the services to have the configuration changes in `core-site.xml` take effect.

Using a Cluster-Dedicated MIT KDC with Active Directory

> **Important:** If you are using Cloudera Manager, ensure you have installed the `openldap-clients` package on the Cloudera Manager Server host before you begin configuring Kerberos authentication.

On the Active Directory Server

1. On the Active Directory server host, type the following command to add the local realm trust to Active Directory:

```
netdom trust HADOOP.EXAMPLE.COM /Domain:EXAMPLE.COM /add /realm /passwordt:TrustPassword
```

2. On the Active Directory server host, type the following command to set the proper encryption type:

   Windows 2003 RC2

Windows 2003 server installations do not support AES encryption for Kerberos. Therefore RC4 should be used. Please see the Microsoft reference documentation for more information.

```
ktpass /MITRealmName HADOOP.EXAMPLE.COM /TrustEncryp RC4
```

Windows 2008

```
ksetup /SetEncTypeAttr HADOOP.EXAMPLE.COM <enc_type>
```

Where the <enc_type> parameter can be replaced with parameter strings for AES, DES, or RC4 encryption modes. For example, for AES encryption, replace <enc_type> with `AES256-CTS-HMAC-SHA1-96` or `AES128-CTS-HMAC-SHA1-96` and for RC4 encryption, replace with `RC4-HMAC-MD5`. See the Microsoft reference documentation for more information.

> **Important:** Make sure that the encryption type you specify is supported on both your version of Windows Active Directory and your version of MIT Kerberos.

On the MIT KDC Server

1. In the `/var/kerberos/krb5kdc/kdc.conf` file on the local dedicated KDC server host, configure the default realm for the Hadoop cluster by substituting your Kerberos realm in the following `realms` property:

```
[realms]
 HADOOP.EXAMPLE.COM = {
```

2. Each of your Hadoop client users must also place this information in their local `core-site.xml` file. The easiest way to do so is by using the Cloudera Manager Admin Console to generate a client configuration file.
3. On the local MIT KDC server host, type the following command in the kadmin.local or kadmin shell to add the cross-realm `krbtgt` principal:

```
kadmin:  addprinc -e "<keysalt_list>" krbtgt/HADOOP.EXAMPLE.COM@EXAMPLE.COM
```

where the <keysalt_list> parameter specifies the types of keys and their salt to be used for encryption of the password for this cross-realm krbtgt principal. It can be set to AES, or RC4 keytypes with a salt value of :normal. Note that DES is deprecated and should no longer be used. You can specify multiple keysalt types using the parameter in the command above. Make sure that at least one of the encryption types corresponds to the encryption types found in the tickets granted by the KDC in the remote realm. For an example of the values to use, see the examples based on the Active Directory functional domain level, below.

Examples by Active Directory Domain or Forest "Functional level"

Active Directory will, based on the Domain or Forest functional level, use encryption types supported by that release of the Windows Server operating system. It is not possible to use AES encryption types with an AD 2003 functional level. If you notice that DES encryption types are being used when authenticating or requesting service tickets to Active Directory then it might be necessary to enable weak encryption types in the `/etc/krb5.conf`. See Sample Kerberos Configuration Files on page 504 for an example.

- **Windows 2003**

```
kadmin: addprinc -e "rc4-hmac:normal" krbtgt/HADOOP.EXAMPLE.COM@EXAMPLE.COM
```

- **Windows 2008**

```
kadmin: addprinc -e "aes256-cts:normal aes128-cts:normal rc4-hmac:normal"
krbtgt/HADOOP.EXAMPLE.COM@EXAMPLE.COM
```

> **Note:** The cross-realm `krbtgt` principal that you add in this step must have *at least one entry* that uses the same encryption type as the tickets that are issued by the remote KDC. If there are no matching encryption types, principals in the local realm can successfully access the Hadoop cluster, but principals in the remote realm are unable to.

On All Cluster Hosts

1. In the `/etc/krb5.conf` file on all cluster hosts and all Hadoop client user hosts, configure both Kerberos realms. Note that `default_realm` should be configured as the local MIT Kerberos realm for the cluster. Your `krb5.conf` may contain more configuration properties than those demonstrated below. This example is provided to clarify configuration parameters. See Sample Kerberos Configuration Files on page 504 for more information.

```
[libdefaults]
  default_realm = HADOOP.EXAMPLE.COM
[realms]
  EXAMPLE.COM = {
    kdc = dc01.example.com:88
    admin_server = dc01.example.com:749
  }
  HADOOP.EXAMPLE.COM = {
    kdc = kdc01.hadoop.example.com:88
    admin_server = kdc01.hadoop.example.com:749
  }
[domain_realm]
  .hadoop.example.com = HADOOP.EXAMPLE.COM
  hadoop.example.com = HADOOP.EXAMPLE.COM
  .example.com = EXAMPLE.COM
  example.com = EXAMPLE.COM
```

2. Use one of the following methods to properly translate principal names from the Active Directory realm into the cluster-dedicated KDC realm for the Hadoop cluster.

   - **Using Cloudera Manager:** Configure the **Trusted Kerberos realms** property of the HDFS service:

     1. Open the Cloudera Manager Admin Console.
     2. Go to the HDFS service.
     3. Click the **Configuration** tab.
     4. Select **Scope** > **HDFS (Service Wide)**
     5. Select **Category** > **Security**.
     6. Type `Kerberos` in the **Search** box.
     7. Edit the **Trusted Kerberos Realms** property to add the name of your central KDC realm. If you need to use more advanced mappings which do more than just allow principals from another domain, you may enter them in the **Additional Rules to Map Kerberos Principals to Short Names** property. For more information about name mapping rules, see Configuring the Mapping from Kerberos Principals to Short Names on page 110.

   - **Using the Command Line:** Configure the `hadoop.security.auth_to_local` setting in the `core-site.xml` file on all of the cluster hosts. The following example translates all principal names with the realm `EXAMPLE.COM` into the first component of the principal name only. It also preserves the standard translation for the default realm (the cluster realm).

```
<property>
  <name>hadoop.security.auth_to_local</name>
  <value>
    RULE:[1:$1@$0](^.*@EXAMPLE\.COM$)s/^(.*)@EXAMPLE\.COM$/$1/g
    RULE:[2:$1@$0](^.*@EXAMPLE\.COM$)s/^(.*)@EXAMPLE\.COM$/$1/g
    DEFAULT
  </value>
</property>
```

Integrating Hadoop Security with Active Directory

Considerations when using an Active Directory KDC

Performance:

As your cluster grows, so will the volume of Authentication Service (AS) and Ticket Granting Service (TGS) interaction between the services on each cluster server. Consider evaluating the volume of this interaction against the Active Directory domain controllers you have configured for the cluster before rolling this feature out to a production environment. If cluster performance suffers, over time it might become necessary to dedicate a set of AD domain controllers to larger deployments. Cloudera recommends you use a dedicated AD instance for every 100 nodes in your cluster. However, note that this recommendation may not apply to high-volume clusters, or cases where the AD host is also being used for LDAP lookups.

Network Proximity:

By default, Kerberos uses UDP for client/server communication. Often, AD services are in a different network than project application services such as Hadoop. If the domain controllers supporting a cluster for Kerberos are not in the same subnet, or they're separated by a firewall, consider using the `udp_preference_limit = 1` setting in the `[libdefaults]` section of the `krb5.conf` used by cluster services. Cloudera strongly recommends *against* using AD domain controller (KDC) servers that are separated from the cluster by a WAN connection, as latency in this service will significantly impact cluster performance.

Process:

Troubleshooting the cluster's operations, especially for Kerberos-enabled services, will need to include AD administration resources. Evaluate your organizational processes for engaging the AD administration team, and how to escalate in case a cluster outage occurs due to issues with Kerberos authentication against AD services. In some situations it might be necessary to enable Kerberos event logging to address desktop and KDC issues within windows environments.

Also note that if you decommission any Cloudera Manager roles or nodes, the related AD accounts will need to be deleted manually. This is required because Cloudera Manager will not delete existing entries in Active Directory.

> **Important:** With CDH 5.1 and higher, clusters managed by Cloudera Manager 5.1 (and higher) do not require a local MIT KDC and are able to integrate directly with an Active Directory KDC. Cloudera recommends you use a direct-to-AD setup. For instructions, see Enabling Kerberos Authentication Using the Wizard on page 50.

If direct integration with AD is not currently possible, use the following instructions to configure a local MIT KDC to trust your AD server:

1. Run an MIT Kerberos KDC and realm local to the cluster and create all service principals in this realm.
2. Set up one-way cross-realm trust from this realm to the Active Directory realm. Using this method, there is no need to create service principals in Active Directory, but Active Directory principals (users) can be authenticated to Hadoop. See Configuring a Local MIT Kerberos Realm to Trust Active Directory on page 174.

Configuring a Local MIT Kerberos Realm to Trust Active Directory

On the Active Directory Server

1. Add the local realm trust to Active Directory with this command:

```
netdom trust YOUR-LOCAL-REALM.COMPANY.COM /Domain:AD-REALM.COMPANY.COM /add /realm
/passwordt:<TrustPassword>
```

2. Set the proper encryption type with this command:

   On Windows 2003 RC2:

```
ktpass /MITRealmName YOUR-LOCAL-REALM.COMPANY.COM /TrustEncryp <enc_type>
```

On Windows 2008:

```
ksetup /SetEncTypeAttr YOUR-LOCAL-REALM.COMPANY.COM <enc_type>
```

The <enc_type> parameter specifies AES, DES, or RC4 encryption. Refer to the documentation for your version of Windows Active Directory to find the <enc_type> parameter string to use.

3. Get and verify the list of encryption types set with this command:

On Windows 2008:

```
ksetup /GetEncTypeAttr YOUR-LOCAL-REALM.COMPANY.COM
```

> **Important:** Make sure the encryption type you specify is supported on both your version of Windows Active Directory and your version of MIT Kerberos.

On the MIT KDC Server

Type the following command in the kadmin.local or kadmin shell to add the cross-realm krbtgt principal. Use the same password you used in the `netdom` command on the Active Directory Server.

```
kadmin:  addprinc -e "<enc_type_list>"
krbtgt/YOUR-LOCAL-REALM.COMPANY.COM@AD-REALM.COMPANY.COM
```

where the <enc_type_list> parameter specifies the types of encryption this cross-realm krbtgt principal will support: either AES, DES, or RC4 encryption. You can specify multiple encryption types using the parameter in the command above, what's important is that at least one of the encryption types corresponds to the encryption type found in the tickets granted by the KDC in the remote realm. For example:

```
kadmin:  addprinc -e "rc4-hmac:normal des3-hmac-sha1:normal"
krbtgt/YOUR-LOCAL-REALM.COMPANY.COM@AD-REALM.COMPANY.COM
```

> **Note:** The cross-realm krbtgt principal that you add in this step must have *at least one entry* that uses the same encryption type as the tickets that are issued by the remote KDC. If no entries have the same encryption type, then the problem you will see is that authenticating as a principal in the local realm will allow you to successfully run Hadoop commands, but authenticating as a principal in the remote realm will not allow you to run Hadoop commands.

On All of the Cluster Hosts

1. Verify that both Kerberos realms are configured on all of the cluster hosts. Note that the default realm and the domain realm should remain set as the MIT Kerberos realm which is local to the cluster.

```
[realms]
  AD-REALM.CORP.FOO.COM = {
    kdc = ad.corp.foo.com:88
    admin_server = ad.corp.foo.com:749
    default_domain = foo.com
  }
  CLUSTER-REALM.CORP.FOO.COM = {
    kdc = cluster01.corp.foo.com:88
    admin_server = cluster01.corp.foo.com:749
    default_domain = foo.com
  }
```

2. To properly translate principal names from the Active Directory realm into local names within Hadoop, you must configure the `hadoop.security.auth_to_local` setting in the `core-site.xml` file on all of the cluster machines. The following example translates all principal names with the realm `AD-REALM.CORP.FOO.COM` into

the first component of the principal name only. It also preserves the standard translation for the default realm (the cluster realm).

```
<property>
  <name>hadoop.security.auth_to_local</name>
  <value>
   RULE:[1:$1@$0](^.*@AD-REALM\.CORP\.FOO\.COM$)s/^(.*)@AD-REALM\.CORP\.FOO\.COM$/$1/g

   RULE:[2:$1@$0](^.*@AD-REALM\.CORP\.FOO\.COM$)s/^(.*)@AD-REALM\.CORP\.FOO\.COM$/$1/g

    DEFAULT
  </value>
</property>
```

For more information about name mapping rules, see Configuring the Mapping from Kerberos Principals to Short Names on page 110.

Integrating Hadoop Security with Alternate Authentication

One of the ramifications of enabling security on a Hadoop cluster is that every user who interacts with the cluster must have a Kerberos principal configured. For some of the services, specifically Oozie and Hadoop (for example, JobTracker and TaskTracker), it can be convenient to run a mixed form of authentication where Kerberos authentication is used for API or command line access while some other form of authentication (for example, SSO and LDAP) is used for accessing Web UIs. Using an alternate authentication deployment is considered an advanced topic because only a partial implementation is provided in this release: you will have to implement some of the code yourself.

> **Note:** The following instructions assume you already have a Kerberos-enabled cluster.

Proceed as follows:

- Configuring the AuthenticationFilter to use Kerberos on page 176
- Creating an AltKerberosAuthenticationHandler Subclass on page 176
- Enabling Your AltKerberosAuthenticationHandler Subclass on page 177

See also the Example Implementation for Oozie on page 178.

Configuring the AuthenticationFilter to use Kerberos

First, you must do all of the steps in the Server Side Configuration section of the Hadoop Auth, Java HTTP SPNEGO Documentation to configure `AuthenticationFilter` to use Kerberos. You must configure `AuthenticationFilter` to use Kerberos before doing the steps below.

Creating an AltKerberosAuthenticationHandler Subclass

An `AuthenticationHandler` is installed on the server-side to handle authenticating clients and creating an `AuthenticationToken`.

1. Subclass the `org.apache.hadoop.security.authentication.server.AltKerberosAuthenticationHandler` class (in the `hadoop-auth` package).
2. When a client sends a request, the authenticate method will be called. For browsers, `AltKerberosAuthenticationHandler` will call the `alternateAuthenticate` method, which is what you need to implement to interact with the desired authentication mechanism. For non-browsers, `AltKerberosAuthenticationHandler` will follow the Kerberos SPNEGO sequence (this is provided for you).
3. The `alternateAuthenticate(HttpServletRequest request, HttpServletResponse response)` method in your subclass should following these rules:
4. Return `null` if the authentication is still in progress; the `response` object can be used to interact with the client.

5. Throw an `AuthenticationException` if the authentication failed.
6. Return an `AuthenticationToken` if the authentication completed successfully.

Enabling Your AltKerberosAuthenticationHandler Subclass

You can enable the alternate authentication on Hadoop Web UIs, Oozie Web UIs, or both. You will need to include a JAR containing your subclass on the classpath of Hadoop or Oozie. All Kerberos-related configuration properties will still apply.

Enabling Your AltKerberosAuthenticationHandler Subclass on Hadoop Web UIs

1. Stop Hadoop by running the following command on every node in your cluster (as root):

```
$ for x in `cd /etc/init.d ; ls hadoop-*` ; do sudo service $x stop ; done
```

2. Set the following property in `core-site.xml`, where `org.my.subclass.of.AltKerberosAuthenticationHandler` is the classname of your subclass:

```
<property>
  <name>hadoop.http.authentication.type</name>
  <value>org.my.subclass.of.AltKerberosAuthenticationHandler</value>
</property>
```

3. (Optional) You can also specify which user-agents you do not want to be considered as browsers by setting the following property as required (default value is shown). Note that all Java-based programs (such as Hadoop client) will use `java` as their user-agent.

```
<property>
  <name>hadoop.http.authentication.alt-kerberos.non-browser.user-agents</name>
  <value>java,curl,wget,perl</value>
</property>
```

4. Copy the JAR containing your subclass into `/usr/lib/hadoop/lib/`.
5. Start Hadoop by running the following command:

```
$ for x in `cd /etc/init.d ; ls hadoop-*` ; do sudo service $x start ; done
```

Enabling Your AltKerberosAuthenticationHandler Subclass on Oozie Web UI

> **Note:**
>
> These instructions assume you have already performed the installation and configuration steps in Oozie Security Configuration.

1. Stop the Oozie Server:

```
sudo /sbin/service oozie stop
```

2. Set the following property in `oozie-site.xml`, where `org.my.subclass.of.AltKerberosAuthenticationHandler` is the classname of your subclass:

```
<property>
  <name>oozie.authentication.type</name>
  <value>org.my.subclass.of.AltKerberosAuthenticationHandler</value>
</property>
```

3. (Optional) You can also specify which user-agents you do not want to be considered as browsers by setting the following property as required (default value is shown). Note that all Java-based programs (such as Hadoop client) will use `java` as their user-agent.

```
<property>
  <name>oozie.authentication.alt-kerberos.non-browser.user-agents</name>
  <value>java,curl,wget,perl</value>
</property>
```

4. Copy the JAR containing your subclass into `/var/lib/oozie`.
5. Start the Oozie Server:

```
sudo /sbin/service oozie start
```

Example Implementation for Oozie

> **Warning:**
>
> The example implementation is **NOT SECURE**. Its purpose is to be as simple as possible, as an example of how to write your own `AltKerberosAuthenticationHandler` subclass.
>
> **It should NOT be used in a production environment**

An example implementation of `AltKerberosAuthenticationHandler` is included (though not built by default) with Oozie. Also included is a simple Login Server with two implementations. The first one will authenticate any user who is using a username and password that are identical, such as `foo:foo`. The second one can be configured against an LDAP server to use LDAP for authentication.

You can read comprehensive documentation on the example at [Creating Custom Authentication](#).

> **Important:**
>
> If you installed Oozie from the CDH packages and are deploying `oozie-login.war` alongside `oozie.war`, you will also need to run the following commands after you copy the `oozie-login.war` file to `/usr/lib/oozie/oozie-server` (if using YARN or `/usr/lib/oozie/oozie-server-0.20` if using MRv1) because it does not automatically be expanded:
>
> ```
> jar xvf oozie-login.war
> mkdir oozie-login
> mv META-INF oozie-login/
> mv WEB-INF oozie-login/
> ```

Authenticating Kerberos Principals in Java Code

This topic provides an example of how to authenticate a Kerberos principal in a Java application using the `org.apache.hadoop.security.UserGroupInformation` class.

The following code snippet authenticates the `cloudera` principal using the `cloudera.keytab` file:

```
// Authenticating Kerberos principal
System.out.println("Principal Authentication: ");
final String user = "cloudera@CLOUDERA.COM";
final String keyPath = "cloudera.keytab";
UserGroupInformation.loginUserFromKeytab(user, keyPath);
```

Data in Transit Encryption (TLS/SSL)

Transport Layer Security (TLS) is an industry standard set of cryptographic protocols for securing communications over a network. TLS evolved from Secure Sockets Layer (SSL), which remains part of the name for historical reasons. This overview includes these topics:

TLS/SSL and Its Use of Certificates

TLS/SSL provides privacy and data integrity between applications communicating over a network by encrypting the packets transmitted between endpoints (ports on a host, for example). Configuring TLS/SSL for any system typically involves creating a *private key* and *public key* for use by server and client processes to negotiate an encrypted connection at runtime. In addition, TLS/SSL can use *certificates* to verify the trustworthiness of keys presented during the negotiation to prevent spoofing and mitigate other potential security issues.

Setting up Cloudera clusters to use TLS/SSL requires creating private key, public key, and storing these securely in a keystore, among other tasks. Although adding a certificate to the keystore may be the last task in the process, the lead time required to obtain a certificate depends on the type of certificate you plan to use for the cluster.

Certificates Overview

A certificate is digitally signed, typically by a *certificate authority* (CA) that indirectly (through a chain of trust) verifies the authenticity of the private key presented during the negotiation. Certificates can be signed in one of the three different ways shown in the table:

Type	Usage Note
Public CA-signed certificates	**Recommended**. This type of certificate is signed by a public certificate authority (CA), such as Symantec or Comodo. Public CAs are trusted third-parties whose certificates can be verified through publicly accessible chains of trust. Using this type of certificate can simplify deployment because security infrastructure, such as root CAs, are already contained in the Java JDK and its default truststore. See Obtain and Deploy Server Certificate on page 185 for details.
Internal CA-signed certificates	This type of certificate is signed by your organization's internal CA. Organizations using OpenSSL Certificate Authority, Microsoft Active Directory Certificate Service, or another internal CA system can use this type of certificate. See How to Configure TLS Encryption for Cloudera Manager for details about using internal CA-signed certificates for TLS Level 3 configuration.
Self-signed certificates	Not recommended for production deployments. Self-signed certificates are acceptable for use in non-production deployments, such as for proof-of-concept setups. See Using Self-signed Certificates for TLS for details.

During the process of configuring TLS/SSL for the cluster, you typically obtain a certificate for each host in the cluster, and re-use the certificate obtained in a given format (JKS, PEM) as needed for the various services (daemon roles) supported by the host. For information about converting formats, see How to Convert Certificate Encodings (DER, JKS, PEM) for TLS/SSL. As an alternative to creating discrete certificates for each host in the cluster, as of Cloudera Manager/CDH 5.9, all Cloudera cluster components support wildcard domains and SubjectAlternateName certificates.

Wildcard Domain Certificates and SAN Certificates Support

Cloudera Manager and CDH (as of release 5.9) support use of wildcard domain certificates and SAN certificates.

A wildcard certificate—a certificate with the common name *****, as in `*.example.com`, rather than a specific host name—can be used for any number of first level sub-domains within a single domain name. For example, a wildcard certificate can be used with `host-1.example.com`, `host-2.example`, `host-3.example.com`, and so on.

Certificates obtained from public CAs are not free, so using wildcard certificates can reduce costs. Using wildcard certificates also makes it easier to enable encryption for transient clusters and for clusters that need to expand and shrink, since the same certificate and keystore can be re-used.

> **Important:** Be aware that using wildcard domain certificates has some security risks. Specifically, because all nodes use the same certificate, a breach of any one machine can result in a breach of all machines.

A SubjectAlternativeName or SAN certificate is a certificate that uses the SubjectAlternativeName extension to associate the resulting certificate with multiple specific host names. In the context of clusters, SAN certificates are used in high-availability (HA) configurations in which a load balancer targets two different specific hosts as primary and secondary nodes for fail-over purposes. See Server Certificate Requirements for HA Deployments for an example.

Wildcard Certificates	Wildcard certificates can be used by all hosts within a given domain. Using wildcard certificates for all hosts in the cluster can reduce costs but also exposes greater potential risk.
SubjectAlternativeName Certificates	SubjectAlternativeName (SAN) certificates are bound to a set of specific DNS names. A single SAN certificate can be used for all hosts or a subset of hosts in the cluster. SAN certificates are used in Cloudera Manager high-availability (HA) configurations.

Obtain Renewed Certificates Before Expiration Dates

The signed certificates you obtain from a public CA (or those you obtain from an internal CA) have an expiration date, such as that shown in this excerpt:

```
$ openssl x509 -in cacert.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 11485830970703032316 (0x9f65de69ceef2ffc)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=MD, L=Baltimore, CN=Test CA/emailAddress=test@example.com
        Validity
            Not Before: Jan 24 14:24:11 2017 GMT
            Not After : Feb 23 14:24:11 2018 GMT
        Subject: C=US, ST=MD, L=Baltimore, CN=Test CA/emailAddress=test@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:b1:7f:29:be:78:02:b8:56:54:2d:2c:ec:ff:6d:
                    ...
                    39:f9:1e:52:cb:8e:bf:8b:9e:a6:93:e1:22:09:8b:
                    59:05:9f
                Exponent: 65537 (0x10001)
```

Expired certificates quite simply do not work and will cause the cluster to fail. Cloudera Manager Agent hosts, for example, will not be able to validate the Cloudera Manager Server host and will fail to launch the cluster nodes. Administrators should note expiration dates of all certificates when they deploy the certificates to the cluster nodes and setup reminders to allow enough time to renew.

Tip: Use OpenSSL to check the expiration dates for certificates already deployed:

```
openssl x509 -enddate -noout -in /opt/cloudera/security/pki/$(hostname -f)-server.cert.pem
```

Understanding Keystores and Truststores

Configuring Cloudera Manager Server and cluster components to use TLS/SSL requires obtaining keys, certificates, and related security artifacts. The following provides a brief overview.

Java Keystore and Truststore

All clients in a Cloudera Manager cluster configured for TLS/SSL need access to the truststore to validate certificates presented during TLS/SSL session negotiation. The certificates assure the client or server process that the issuing authority for the certificate is part of a legitimate chain of trust.

The standard Oracle Java JDK distribution includes a default **truststore** (`cacerts`) that contains root certificates for many well-known CAs, including Symantec. Rather than using the default truststore, Cloudera recommends using the **alternative truststore**, `jssecacerts`. The alternative truststore is created by copying `cacerts` to that filename (`jssecacerts`). Certificates can be added to this truststore when needed for additional roles or services. This alternative truststore is loaded by Hadoop daemons at startup.

The private keys are maintained in the **keystore**.

Note: For detailed information about the Java keystore and truststore, see Oracle documentation:

- Keytool—Key and Certificate Management Tool
- JSSE Reference Guide for Java

Although the keystore and truststore in some environments may comprise the same file, as configured for Cloudera Manager Server and CDH clusters, the keystore and truststore are distinct files. For Cloudera Manager Server clusters, each host should have its own keystore, while several hosts can share the same truststore. This table summarizes the general differences between keystore and the truststore in Cloudera Manager Server clusters.

Keystore	Truststore
Used by the server side of a TLS/SSL client-server connection.	Used by the client side of a TLS/SSL client-server connection.
Typically contains 1 private key for the host system.	Contains no keys of any kind.
Contains the certificate for the host's private key.	Contains root certificates for well-known public certificate authorities. May contain certificates for intermediary certificate authorities.
Password protected. Use the same password for the key and its keystore.	Password-protection not needed. However, if password has been used for the truststore, never use the same password as used for a key and keystore.
Password stored in a plaintext file read permissions granted to a specific group only (OS filesystem permissions set to 0440, `hadoop:hadoop`).	Password (if there is one for the truststore) stored in a plaintext file readable by all (OS filesystem permissions set to 0440).
No default. Provide a keystore name and password when you create the private key and CSR for any host system.	For Java JDK, **cacerts** is the default unless the alternative default **jssecacerts** is available.
Must be owned by hadoop user and group so that HDFS, MapReduce, YARN can access the private key.	HDFS, MapReduce, and YARN need client access to truststore.

The details in the table above are specific to the Java KeyStore (JKS) format, which is used by Java-based cluster services such as Cloudera Manager Server, Cloudera Management Service, and many (but not all) CDH components and services. See Certificate Formats (JKS, PEM) and Cluster Components on page 182 for information about certificate and key file type used various processes.

CDH Services as TLS/SSL Servers and Clients

Cluster services function as a TLS/SSL server, client, or both:

Component	Client	Server
HBase	⊘	✓
HDFS	✓	✓
Hive	✓	✓
Hue (Hue is a TLS/SSL client of HDFS, MapReduce, YARN, HBase, and Oozie.)	✓	⊘
MapReduce	✓	✓
Oozie	⊘	✓
YARN	✓	✓

Daemons that function as TLS/SSL servers load the keystores when starting up. When a client connects to an TLS/SSL server daemon, the server transmits the certificate loaded at startup time to the client, and the client then uses its truststore to validate the certificate presented by the server.

Certificate Formats (JKS, PEM) and Cluster Components

Cloudera Manager Server, Cloudera Management Service, and many other CDH services use JKS formatted keystores and certificates. Cloudera Manager Agent, Hue, Key Trustee Server, Impala, and other Python or C++ based services require PEM formatted certificates and keystores rather than Java. Specifically, PEM certificates conform to PKCS #8, which requires individual Base64-encoded text files for certificate and password-protected private key file. The table summarizes certificate types required by several components.

Component	JKS	PEM
HBase	✓	⊘
HDFS	✓	⊘
Hive (Hive clients and HiveServer 2)	✓	⊘
Hue	⊘	✓
Impala	⊘	✓
MapReduce	✓	⊘
Oozie	✓	⊘
Solr	⊘	✓
YARN	✓	⊘

For more information, see:

- How to Convert Certificate Encodings (DER, JKS, PEM) for TLS/SSL
- OpenSSL Cryptography and TLS/SSL Toolkit

Recommended Keystore and Truststore Configuration

Cloudera recommends the following for keystores and truststores for Cloudera Manager clusters:

- Create a separate keystore for each host. Each keystore should have a name that helps identify it as to the type of host—server or agent, for example. The keystore contains the private key and should be password protected.
- Create a single truststore that can be used by the entire cluster. This truststore contains the root CA and intermediate CAs used to authenticate certificates presented during TLS/SSL handshake. The truststore does not need to be password protected. (See How To Add Root and Intermediary CAs to Truststore for more information about the truststore for TLS/SSL and Cloudera clusters.)

The steps included in Obtain and Deploy Server Certificate on page 185 follow this approach.

Configuring Cloudera Manager Clusters for TLS/SSL

As discussed in Data in Transit Encryption, TLS/SSL is a security protocol designed to prevent eavesdropping, tampering, and message forgery by encrypting network communications. It also supports authentication of host certificates prior to encryption, to prevent spoofing.

Cloudera Levels for TLS/SSL Support

Three increasingly secure TLS levels defined by Cloudera are shown in the table below. These levels have been defined by Cloudera to simply break up TLS/SSL configuration tasks into manageable tasks. Each level provides incrementally stricter security for the cluster.

Level	Description and configuration process
Level 1 (Minimal)	Encrypted communications between a Web browser and Cloudera Manager, and between Agents and Cloudera Manager. This level encrypts connections between a Web browser running the Cloudera Manager Admin Console and the Cloudera Manager Server. • Level 0: Basic TLS/SSL Configuration on page 184 • Level 1: Enabling Encryption for the Cluster on page 190
Level 2 (Better)	Encrypted communications (as with Level 1) *plus* Agents verify authenticity of Cloudera Manager Server's TLS certificate. • Complete all Level 1 configuration processes. • Level 2: Enabling Cloudera Manager Agent Hosts to Authenticate the Server's Certificate on page 192
Level 3 (Best)	Encrypted communications (as with Level 1) and Cloudera Manager Server certificate presentation (as with Level 2), *plus* each Agent presents a certificate to Cloudera Manager Server to prevent spoofing by untrusted Agents running on hosts. • Complete all Level 1 and Level 2 configuration processes. • Level 3: Configuring the Cluster to Authenticate Agent Certificates on page 193 For a start-to-finish configuration guide for Level 3 TLS, see How to Configure TLS Encryption for Cloudera Manager.

As shown in the table, TLS levels are cumulative—Level 1 must be configured before Level 2, and Level 2 must be configured Level 3.

TLS/SSL Configuration and Kerberos Integration

Cloudera recommends that clusters deployed to production be configured using both TLS/SSL for wire (data in transit) encryption and using Kerberos for authentication. Because Cloudera Manager distributes keytabs to the cluster nodes during the process of integrating the cluster with the Kerberos instance (specifically, with the KDC or "Key Distribution Center"), Cloudera recommends that you configure TLS Level 1 for the cluster *before* integrating Kerberos. This approach ensures that if any keytabs are intercepted, they will not be readable. The recommended sequence is as follows:

1. Configure TLS/SSL for encryption (through Level 1):

- Level 0: Basic TLS/SSL Configuration on page 184
- Level 1: Enabling Encryption for the Cluster on page 190

2. Integrate the cluster with your organization's Kerberos deployment:

- Enabling Kerberos Authentication Using the Wizard on page 50

3. Continue configuring TLS/SSL for certificate authentication (Level 2, Level 3):

- Level 2: Enabling Cloudera Manager Agent Hosts to Authenticate the Server's Certificate on page 192
- Level 3: Configuring the Cluster to Authenticate Agent Certificates on page 193

Plan ahead for the Kerberos integration as part of the TLS/SSL configuration if that is your goal for the cluster. To integrate the cluster with your Kerberos or Active Directory, you must have admin privileges on those systems or help from your organization's Kerberos or Active Directory administrator for that part of the process.

Consider setting up a complete Cloudera Manager cluster without TLS/SSL, unless you have experience with both clusters and TLS/SSL.

> **Note:** If Kerberos has not been integrated with the cluster, the Cloudera Manager Admin Console displays warning text on several of the TLS/SSL configuration items.

Recommendations

In addition to configuration sequence outlined in TLS/SSL Configuration and Kerberos Integration on page 183, Cloudera recommends the following:

- Configure production clusters for TLS Level 3. This is the most secure form of TLS because it authenticates not only the Cloudera Manager Server host but also Cloudera Manager Agent host system certificates before encrypting the communications.
- Enable TLS/SSL for all services running on the cluster whenever you enable TLS/SSL for any one service. This is especially important for services that work together to process data. For example, if the cluster supports HDFS, MapReduce, and YARN and TLS/SSL has been enabled for the HDFS service, TLS/SSL must be enabled for MapReduce and YARN as well.
- Configure all clusters managed by any given Cloudera Manager instance for the same TLS level. Cloudera recommends that all clusters managed by a single Cloudera Manager instance have comparable security requirements. Do not manage production clusters, test clusters, and development clusters—all of which likely have different security requirements—from the same Cloudera Manager instance.

Level 0: Basic TLS/SSL Configuration

Configuring a Cloudera Manager cluster to use TLS for encrypted network and intra-cluster communications is a multi-step process involving many tasks, including using Linux shell commands on the Cloudera Manager Server host system, configuring the Cloudera Manager Agent host's configuration files, and using the Cloudera Manager Admin Console to enable TLS/SSL capabilities. Completing these tasks requires that you have:

- Privileges as user `root` (able to sudo) on the hosts that comprise the cluster;
- Cloudera Manager Admin Console role of **Cluster Administrator** or **Full Administrator**.

Cloudera Management Service and TLS/SSL

Configuring TLS/SSL on any server affects how clients interact with that server. For browsers, which communicate over HTTP, TLS/SSL configured on a server host redirects traffic from the HTTP port (7180) to the secure HTTP port, HTTPS (7183). When TLS Level 0 configuration is complete, the Cloudera Management Service roles are enabled for TLS encryption. Similarly, RPC clients are redirected to their secure port.

Cloudera Management Service Roles and HTTPS Communications

Cloudera Management Service is transparently installed during the Cloudera Management Server installation. It is a service available from the Cloudera Manager Admin Console that comprises the monitoring and reporting roles shown in the table below.

HTTPS Client	Web servers (HTTPS Service)					
Role	**Cloudera Manager Server**	**Name Node**	**Job Tracker**	**Oozie**	**Impala**	**YARN**
Activity Monitor	✔	⊘	✔	✔	⊘	⊘
Host Monitor	✔	⊘	⊘	⊘	⊘	⊘
Service Monitor	✔	✔	⊘	✔	✔	✔
Event Server	✔	⊘	⊘	⊘	⊘	⊘
Reports Manager	✔	✔	⊘	⊘	⊘	⊘

When the cluster starts, these Cloudera Management Service roles connect to the Cloudera Manager Server and access the truststore to validate the Cloudera Manager Server's certificate and complete the TLS/SSL connection.

Level 0 is comprises the preliminary tasks that will be used in subsequent levels. The tasks include:

Obtain and Deploy Server Certificate

Cloudera recommends using certificates obtained from one of the trusted public certificate authorities (CA) when configuring Cloudera Manager clusters for TLS/SSL encryption. The steps below show you how to:

- Create digital keys
- Create a certificate signing request (CSR)
- Submit CSR to certificate authority (CA)
- Load the certificate into the server keystore
- Create a truststore for specific use by your cluster

> **Note:** If your organization uses its own internal CA, see How to Configure TLS Encryption for Cloudera Manager instead of following steps in this section. How to Configure TLS Encryption for Cloudera Manager is a start-to-finish guide for configuring clusters for TLS level 3 using certificates from an internal—rather than a public—CA.

Certificate and Key Management Utilities Overview

Keys, keystores, certificates, and other security artifacts can be created and managed using key management utilities, including the Java Keytool and OpenSSL. In addition to the brief overviews below, see How to Convert PEM to JKS and JKS to PEM for TLS/SSL Services and Clients for more information about using these two tools.

Java Keytool

Oracle Java `keytool` is a utility included with the Oracle JDK for creating and managing cryptographic keys and certificates. During configuring the Cloudera Manager Cluster for TLS/SSL, you create the private key pairs, keystore, certificate signing requests, and create a truststore for specific use by the cluster using this software tool, as detailed in the steps throughout this guide.

Java Keytool Usage for Cloudera Manager TLS/SSL Configuration

For any steps using the Java Keytool, be sure to:

- Use the Oracle Java `keytool` rather than tools such as OpenJDK.

- Use the JDK downloaded from Oracle or the Cloudera-provided Oracle JDK located in this default path on a Cloudera Manager Server host:

```
/usr/java/jdk1.7.0_67-cloudera/bin/jre/lib/security
```

- Use the same version of the Java keytool for all steps. If the host has multiple JDKs installed, set the PATH variable so that the Oracle JDK is invoked first, as in this example:

```
$ export JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera
$ export PATH=$JAVA_HOME/bin:$PATH
```

- Use the same *password* for the `-keypass` and `-storepass` in any commands that invoke these two options. Cloudera Manager requires the same password for a key and its keystore.

OpenSSL

OpenSSL is an open source cryptography and TLS/SSL toolkit that has been widely used since its inception ~ 1999. Just as with Java Keytool, OpenSSL lets you create private keys, certificate requests, and keystores, and it provides options for verifying certificates.

Cloudera Manager Agent hosts act as clients of a Cloudera Manager Server host, as in RPC client and server communications. The agent hosts require PEM certificates (you create them later in the process, as part of Level 3: Configuring the Cluster to Authenticate Agent Certificates on page 193). Hue, Impala, and other Python-based services also use PEM-formatted certificates and keys. Use OpenSSL to convert certificates from one format to the other. See How to Convert PEM to JKS and JKS to PEM for TLS/SSL Services and Clients for details.

> **Note:** The steps below assume certificates obtained from a public Certificate Authority (CA).

Step 1: Create Directory for Security Artifacts

Distributing the certificates, keys, truststore—in short, all security artifacts used for TLS/SSL intra-cluster communications—is part of this and some subsequent processes. To keep things organized, Cloudera recommends that you create the directory and distribute the artifacts when you receive them, even though they may not be immediately needed.

The following table shows the recommended directory structure for the security artifacts created in this and subsequent sections. Use your own names if you prefer but for ease of deployment use the same directory names across all hosts in the cluster.

Example	Description
`cmhost.sec.example.com`	FQDN of an example Cloudera Manager Server host.
`/opt/cloudera/security`	Base path for security-related files.
`/opt/cloudera/security/pki`	Path for all security artifacts associated with TLS/SSL, including keys, keystores (*`keystore`*`.jks`), CSR, and root- and intermediate-CA certificates.
`/usr/java/jdk1.7.0_67-cloudera/jre/lib/security/jssecacerts`	Path to the default alternative Java truststore on a Cloudera Manager Server host system.

1. On the Cloudera Manager Server host, create the `/opt/cloudera/security/pki` directory:

```
$ sudo mkdir -p /opt/cloudera/security/pki
```

2. This directory must be owned by `cloudera-scm:cloudera-scm` and have permissions set correctly (executable bit for user:group) so that Cloudera Manager can access the keystore at runtime:

```
$ sudo chown -R cloudera-scm:cloudera-scm /opt/cloudera/security/pki
$ umask 077
$ umask 027
$ cd /opt/cloudera/security/pki
```

Directories and artifacts persist during system upgrades. For security purposes, for any host you remove from a cluster, remove any directories you create and more importantly, remove any security artifacts (keys, certificates, and so on) they may contain.

Step 2: Create the Java Truststore

On the Cloudera Manager Server host, copy the JDK `cacerts` file to `jssecacerts`:

```
$ sudo cp $JAVA_HOME/jre/lib/security/cacerts $JAVA_HOME/jre/lib/security/jssecacerts
```

> **Note:** For certificates obtained from an internal (rather than a public) CA, you must import the CA. See How to Add Root and Intermediate CAs to Truststore for TLS/SSL on page 446 for details.

If you do not have the `$JAVA_HOME` variable set, replace it with the path to the Oracle JDK. For example, the default path to the Java JDK on a Cloudera Manager Server host is:

`/usr/java/jdk1.7.0_67-cloudera/`

Step 3: Generate Server Key and CSR

The steps in this section generate discrete public keys and create CSRs to obtain the certificate for each. To create a SAN certificate such as for an HA configuration with a load balancer, see Server Certificate Requirements for HA Deployments for an example.

> **Note:** Always check with your selected public CA **before** creating any CSRs, and be sure to follow the CA-specific processes. For an internal CA, follow your organization's internal process for creating and submitting CSRs.

1. On the Cloudera Manager Server host, use the `keytool` utility to generate a keypair and a keystore named for the server. Replace the `OU`, `O`, `L`, `ST`, and `C` entries with the values for your organization name, location, and country code (US, in the example):

```
$ sudo keytool -genkeypair -alias $(hostname -f)-server -keyalg RSA -keystore \
/opt/cloudera/security/pki/$(hostname -f)-server.jks -keysize 2048 -dname \
"CN=$(hostname -f),OU=Dept,O=Example.com,L=City,ST=State,C=US" \
-storepass password -keypass password
```

   Use the same *password* for the key and its keystore (`-keypass` and `-storepass`, respectively): Cloudera Manager does not support using different passwords for the key and keystore.

2. Generate a certificate signing request (CSR) for the public key (contained in the keystore as a result of the command above). In this command below, enter the password that you set for the key and the keystore in the command above:

```
$ sudo keytool -certreq -alias $(hostname -f)-server \
-keystore /opt/cloudera/security/pki/$(hostname -f)-server.jks \
-file /opt/cloudera/security/pki/$(hostname -f)-server.csr -storepass password \
-keypass password
```

Step 4: Submit the CSR to the CA

1. Submit the CSR file to your certificate authority using the process and means required by the CA, for example, email or web submission. For the certificate format, specify PEM (base64 ASCII) (see Step 5 below for an example of PEM formatted certificate heading and closing structure).
2. The public CA will request specific details from you, to verify that you own the domain name contained in the CSR, before they issue the certificate.
3. The public CA sends you the signed certificate for your public key. The certificate includes (among other details) `notBefore` and `notAfter` properties that specify the lifetime of the certificate, which is typically one-year from the time issued.

> **Important:** Document the expiration dates for all certificates you deploy in your clusters so that you can pro-actively renew certificates prior to their expiration. See Obtain Renewed Certificates Before Expiration Dates on page 180 for details about checking expiration dates on certificates already deployed.

4. When you receive the signed certificate from the CA, you can proceed with Step 5.

Step 5: Verify the Certificate

From the public (or internal) CA, you receive the certificate for the server signed by the CA and several other digital artifacts, including root CA and possibly one (or more) intermediate CA certificates. Before distributing these to the hosts, make sure the certificate is in PEM format. A PEM formatted certificate file looks like this:

```
-----BEGIN CERTIFICATE-----
<The encoded certificate is represented by multiple lines of exactly 64 characters,
except
for the last line, which can contain 64 characters or fewer.>
-----END CERTIFICATE-----
```

If your issued certificate is in binary (DER) format, convert it to PEM format before continuing. See How to Convert Formats (PEM, JKS) for TLS/SSL Clients and Services for details.

To modify the truststore (jssecacerts) to explicitly trust certificates or certificate chain (as you might for certificates signed by an internal CA), follow the steps in How to Add Root and Intermediate CAs to Truststore for TLS/SSL.

Step 6: Import the Certificate into the Keystore

Copy the signed certificate you receive from the CA to the Cloudera Manager Server host. To identity the certificate's functionality, include a suffix such as "-server.cert.pem" in the target name for the copy command, as shown in this example:

```
cp cert-file-recd /opt/cloudera/security/pki/$(hostname -f)-server.cert.pem
```

Import the certificate into the keystore.

```
$ sudo keytool -importcert -alias $(hostname -f)-server \
-file /opt/cloudera/security/pki/$(hostname -f)-server.cert.pem \
-keystore /opt/cloudera/security/pki/$(hostname -f)-server.jks
```

Assuming the certificate was obtained from a public CA, you can safely disregard this message about trust, and enter yes to continue:

```
... is not trusted. Install reply anyway? [no]:  yes
```

You *must* see the following response confirming that the certificate has been properly imported and can verify the private key that it certifies.

```
Certificate reply was installed in keystore
```

If you do not see this response, double-check all your steps up to this point:

- Verify that you are using the correct paths
- Verify that you are using the appropriate certificate
- Check all your steps to this point

If you cannot successfully import the certificates, see Getting Support for information about resources for help, including how to contact Cloudera Support.

Enable TLS/SSL Encryption for Cloudera Manager Admin Console

Required Role: **Cluster Administrator** or **Full Administrator**

These tasks require you to access the Cloudera Manager Admin Console. These steps assume you have completed Step 1 through Step 6 to Obtain and Deploy Server Certificate on page 185. After you complete the steps below, the Cloudera Manager Admin Console is accessible over the HTTPS port, and that the Cloudera Management Service roles can connect to Cloudera Manager Server at startup. Some of the settings also prepare the way for subsequent TLS/SSL levels.

Step 7: Enable HTTPS for the Cloudera Manager Admin Console

This step ensures that browsers connecting the Cloudera Manager Admin Console use the HTTPS port (7183) and encrypts communications between the browser and the server.

1. Log in to the Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Select the **Security** category.
4. Scroll through the settings to the section for TLS encryption settings and enter values for the following:

Property	Description
Use TLS Encryption for Admin Console	Check the box to enable TLS encryption for Cloudera Manager.
Cloudera Manager TLS/SSL Server JKS Keystore File Location	Enter the complete path to the keystore created in Step 3: Generate Server Key and CSR on page 187. Replace the example path and filename with your own settings. The example path with example filename are as follows: `/opt/cloudera/security/pki/`*`cmsrv.example.com`*`.jks`
Cloudera Manager TLS/SSL Server JKS Keystore File Password	Enter the password for the keystore.

5. Click **Save Changes** to save the settings.

Step 8: Specify TLS/SSL Truststore Properties for Cloudera Management Services

While still logged in to the Cloudera Manager Admin Console:

1. Select **Clusters** > **Cloudera Management Service**.
2. Click the **Configuration** tab.
3. Select **Scope** > **Cloudera Management Service (Service-Wide)**.
4. Select **Category** > **Security**.
5. Enter values for the following TLS/SSL properties:

Property	Description
TLS/SSL Client Truststore File Location	Enter the path to the truststore you created for this cluster in Step 2: Create the Java Truststore on page 187: `$JAVA_HOME/jre/lib/security/jssecacerts` If you leave this field empty, certificates are verified using the default Java truststore, `cacerts`.

Property	Description
TLS/SSL Client Truststore File Password	Enter the password for the truststore file. If you created `jssecaerts` from `cacerts`, the default password is `changeit`.

6. Click **Save Changes** to save the settings.

You must restart both Cloudera Manager Server and the Cloudera Management Service for the system to implement these changes, and to enable the Cloudera Management Service roles (Host Monitor, Service Monitor, and so on) to communicate with Cloudera Manager Server.

Step 9: Restart Cloudera Manager and Services

1. Restart the Cloudera Manager Server by running `service cloudera-scm-server restart` on the Cloudera Manager Server host.
2. After the restart completes, connect to the Cloudera Manager Admin Console:

```
https://cm01.example.com:7183
```

For server certificates signed by an internal CA, configure the browser to explicitly trust the certificate, to avoid seeing the warning prompt each time you connect to Cloudera Manager Admin Console.

From the Cloudera Manager Admin Console:

1. Select **Clusters** > **Cloudera Management Service**.
2. From the Cloudera Management Service **Actions** drop-down menu, select **Restart**.

At the conclusion of Level 0 tasks, the Cloudera Manager cluster is configured for encryption only, between:

- Browsers and the Cloudera Manager Admin Console
- Cloudera Management Service roles and Cloudera Manager Server

Level 1: Enabling Encryption for the Cluster

Required Role: Cluster Administrator or **Full Administrator**

Level 1 TLS ensures that the cluster encrypts network traffic between client and server processes within the cluster, specifically between Cloudera Manager Agent hosts and the Cloudera Manager Server. This level is encryption only: Certificates are neither presented nor verified as a result of this process.

Prerequisites

The steps below assume that basic TLS configuration has been completed, specifically:

- Level 0: Basic TLS/SSL Configuration on page 184

Step 1: Enable TLS Encryption for the Cloudera Manager Agents

In this step, you enable TLS properties for Cloudera Manager Agents and their connections to the Cloudera Manager Server.

1. Log into the Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Click the **Security** category.
4. Configure the following TLS settings in the Cloudera Manager Server:

Property	Description
Use TLS Encryption for Agents	Check the box to encrypt communications between Cloudera Manager Server and Cloudera Manager Agent hosts.

5. Click **Save Changes**.

Step 2: Modify the Cloudera Manager Agent Configuration File

Each Cloudera Manager Agent host system in the cluster needs to have the `use_tls` property in its configuration file set to 1, to enable TLS (the default is 0).

1. Use a text editor to open the configuration file, located in this path on a Cloudera Manager Agent host:

```
/etc/cloudera-scm-agent/config.ini
```

2. Find the [Security] section in the file and change the value of `use_tls` to 1, as shown below:

```
[Security]
# Use TLS and certificate validation when connecting to the CM server.
use_tls=1
```

3. Apply this change to each Cloudera Manager Agent host's configuration file. You can do this in one of two ways:

    - Modify each host's configuration file, as above;
    - Modify one host's configuration file and then copy that file to all other hosts in the cluster. Take this approach only if none of the hosts has customizations, such as changes from the default `listening_hostname` or `listening_ip address`, in the configuration file. By default, `config.ini` has no host-specific details.

Step 3: Restart the Cloudera Manager Server

On the Cloudera Manager Server host:

Launch a terminal session and use the command line to restart the server and activate the TLS configuration, as shown below:

```
$ sudo service cloudera-scm-server restart
```

Step 4: Restart the Cloudera Manager Agents

On each Cloudera Manager Agent host:

Restart the Cloudera Manager Agent daemon as shown below:

```
$ sudo service cloudera-scm-agent restart
```

Step 5: Check Cloudera Manager Server-Agent Communications

You can observe the start-up processes as they occur throughout the cluster, using the Cloudera Manager Admin Console, and you can check the status of the cluster's heartbeat to confirm successful communications.

1. Open the Cloudera Manager Admin Console.
2. Select **Hosts** > **All Hosts**.
3. Open the **Last Heartbeat** filter to see its status. This status should be **Good**, meaning that the server and agent hosts are communicating successfully.

If the Last Heartbeat appears to have failed, you can check for error messages in the Cloudera Manager Agent log. See TLS Level 2, Step 4 for details about accessing the log.

Step 6: Integrate Kerberos (Optional)

As discussed in TLS/SSL Configuration and Kerberos Integration on page 183, if you plan to use Kerberos (MIT Kerberos, Microsoft Active Directory) for user and service authentication for the cluster, this is the point at which to integrate your organization's Kerberos service. See Enabling Kerberos Authentication Using the Wizard on page 50 for details.

Although Kerberos integration is optional (and not a requirement for configuring TLS/SSL), Cloudera recommends integrating the cluster with an organization's Kerberos instance (MIT Kerberos, Active Directory).

Level 2: Enabling Cloudera Manager Agent Hosts to Authenticate the Server's Certificate

Required Role: Cluster Administrator or **Full Administrator**

Level 2 TLS ensures that all Cloudera Manager Agent host systems check the validity of the certificate presented by Cloudera Manager Server during the TLS handshake. If the certificate is missing or expired, or if its authenticity cannot be verified through the associated CA chain, the agent aborts the connection, thus ensuring that Cloudera Manager Agent hosts are not spoofed by a bogus server.

Prerequisites

The steps below assume that Level 1 TLS configuration has been completed, specifically:

- Level 0: Basic TLS/SSL Configuration on page 184
- Level 1: Enabling Encryption for the Cluster on page 190

> **Note:** For certificates signed by an internal CA, you must explicitly trust the CA before proceeding. See How to Add Root and Intermediate CAs to Truststore for TLS/SSL for details.

Step 1: Modifying the Cloudera Manager Agent Configuration File

Each Cloudera Manager Agent host system in the cluster needs to have the location of the Cloudera Manager Server certificate set for the `verify_cert_file` property.

1. Use a text editor to open the configuration file, located in this path:

```
/etc/cloudera-scm-agent/config.ini
```

2. In the [Security] section, find this text:

```
[Security]
...
# verify_cert_file=/opt/cloudera/security/pki/rootca.cert.pem
```

3. Apply this change to each Cloudera Manager Agent host's configuration file. You can do this in one of two ways:
   - Modify each host's configuration file;
   - Modify one host's configuration file and then copy that file to all other hosts in the cluster. Take this approach only if none of the hosts has customizations already in the configuration file, such as changes from the default `listening_hostname` or `listening_ip address`. By default, `config.ini` has no host-specific details.

Step 2: Restart the Cloudera Manager Server

On the Cloudera Manager Server host:

Launch a terminal session and use the command line to restart the server and activate the TLS configuration, as shown below:

```
$ sudo service cloudera-scm-server restart
```

Step 3: Restart the Cloudera Manager Agents

On each Cloudera Manager Agent host:

Restart the Cloudera Manager Agent daemon as shown below:

```
$ sudo service cloudera-scm-agent restart
```

Step 4: Check Cloudera Manager Server-Agent Communications

You can observe the start-up processes as they occur throughout the cluster, using the Cloudera Manager Admin Console, and you can check the status of the cluster's heartbeat to confirm successful communications.

1. Open the Cloudera Manager Admin Console.
2. Select **Hosts** > **All Hosts**.
3. Open the **Last Heartbeat** filter to see its status. This status should be **Good**, meaning that the server and agent hosts are communicating successfully.

If the Last Heartbeat appears to have failed, you can check for error messages in the Cloudera Manager Agent log, located by default in this path:

```
/var/log/cloudera-scm-agent/cloudera-scm-agent.log
```

To access the log file in this path:

```
$ sudo su
# cd /var/log/cloudera-scm-agent
```

The log is accessible from the Cloudera Manager Admin Console as follows:

- Select **Diagnostics** > **Logs**.
- Open the **Select Sources** drop-down selector, and deselect all sources except for **Cloudera Manager** > **Agent**.

Level 3: Configuring the Cluster to Authenticate Agent Certificates

Required Role: Cluster Administrator or **Full Administrator**

Level 3 TLS configuration ensures that each Cloudera Manager Agent host presents a signed certificate to Cloudera Manager Server prior to encryption, thus preventing possible introduction of a rogue host into the cluster. The steps in this section include obtaining signed certificates from a public CA.

Prerequisites

The steps on this page assume that Level 2 TLS has already been completed:

- Level 0: Basic TLS/SSL Configuration on page 184,
- Level 1: Enabling Encryption for the Cluster on page 190, and
- Level 2: Enabling Cloudera Manager Agent Hosts to Authenticate the Server's Certificate on page 192

Preliminary Task: Obtaining Certificates for the Agent

Check with the specific CA you plan to use before generating CSRs and follow their specific process.

You need to obtain certificates for each Cloudera Manager Agent host in the cluster.

Step 1: Create Directories for Security Artifacts

On each Cloudera Manager Agent host, create a directory in which to store the private key and certificates. For example:

```
$ sudo mkdir -p /opt/cloudera/security/pki
```

Use the same directory on all cluster hosts to simplify management and maintenance.

Step 2: Generate Keys and CSRs for Cloudera Manager Agent Hosts

Generate keys and CSRs for each Cloudera Manager Agent host in the cluster using Java keytool. The command-line syntax in the steps below uses the `$(hostname -f)` variable to pass in the name of the host on which the command is run, to simplify the process.

On each Cloudera Manager Agent host:

1. Create a Java keystore and private key for use by the agent host. Use the same password for the key and the keystore (Cloudera Manager requires they be the same) as shown here:

```
$ keytool -genkeypair -alias $(hostname -f)-agent -keyalg RSA -keystore \
/opt/cloudera/security/pki/$(hostname -f)-keystore.jks -keysize 2048 -dname \
"CN=$(hostname -f),OU=Dept,O=Example,L=City,ST=State,C=US" \
-storepass password -keypass password
```

2. Generate the CSR for certificates that can authenticate as both a client and a server, using the extended attributes `serverAuth` and `clientAuth`, as shown here:

```
$ keytool -certreq -alias $(hostname -f)-agent  \
-keystore /opt/cloudera/security/pki/$(hostname -f)-agent.jks  \
-file /opt/cloudera/security/pki/$(hostname -f)-agent.csr  \
-ext EKU=serverAuth,clientAuth  \
-storepass password -keypass password
```

Public CAs may sometimes ignore requested extensions in a CSR, so make sure you specify that you need certificates with both server and client authentication options.

Step 3: Submit the CSR to the CA

1. Submit the CSR file to your certificate authority using the process and means required by the CA, for example, email or web submission.
2. The public CA will request specific details from you, to verify that you own the domain name contained in the CSR, before they issue the certificate.
3. When you receive the signed certificate from the CA, you can proceed with to Step 4.

Step 4: Verify the Certificate

Use OpenSSL tool to confirm that each signed certificate includes both server and client authentication options:

```
$ openssl x509 -in /opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem -noout -text
```

The response should include both `TLS Web Server Authentication` and `TLS Web Client Authentication`, as shown below:

```
  X509v3 Extended Key Usage:
    TLS Web Server Authentication, TLS Web Client Authentication
```

If either value is missing from the certificate's `X509v3 Extended Key Usage` section:

1. Check your CSR and make sure it was created using both extended attribute options,
2. Re-submit the CSR to the CA and re-iterate that the signed certificate must include both options, so that it can authenticate as a server and a client.

Assuming you have properly constructed certificates, you can now distributed the certificates to the Cloudera Manager Agent host comprising the cluster.

Concatenated Certificates and Multiple Intermediate Certificates

If the CA provided you with the root CA and intermediate file concatenated into a single file, split the file along the `END CERTIFICATE/BEGIN CERTIFICATE` boundary into individual files.

If the CA provided you with multiple intermediate CA certificates, use unique file names for each, such as `intca-1.cert.pem`, `intca-2.cert.pem`, and so on.

Use these certificates as needed in Step 5.

Step 5: Distribute the Certificates to the Cloudera Manager Agent Hosts

On each Cloudera Manager Agent host:

1. Copy each signed certificate to the appropriate host in the path shown below. Add a suffix *-agent.cert* to each for easy identification (as to functionality) as shown here:

```
/opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem
```

2. Copy any root and intermediate CA certificates to the following paths:

```
/opt/cloudera/security/pki/rootca.cert.pem
/opt/cloudera/security/pki/intca.cert.pem
```

3. Append the intermediate CA certificate to the signed certificate. Note the append operator (>>) in the command below, not the overwrite operator (>):

```
$ sudo cat /opt/cloudera/security/pki/intca.cert.pem >>
/opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem
```

4. Import the signed certificate into the keystore:

```
$ sudo keytool -importcert -alias $(hostname -f)-agent \
-file /opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem \
-keystore /opt/cloudera/security/pki/$(hostname -f)-agent.jks
```

The command should return a message such as that shown below. Enter `yes` at the prompt to continue:

```
... is not trusted. Install reply anyway? [no]:  yes
```

You should see a message confirming successful import of the signed certificate into the keystore, certifying the private key:

```
Certificate reply was installed in keystore
```

> **Note:** If you do not receive confirmation that the certificate was installed successfully in the keystore, contact Cloudera Support.

Assuming the certificate imported successfully, you can continue.

Step 6: Creating Symbolic Links to Certificate and Keystore

Rather than using individual configuration files on each Cloudera Manager Agent host system, you can use a single file and create symbolic links (symlinks) to it from each of the hosts. This assumes that the hosts can use the same setup, and there are no customizations in the Cloudera Manager Agent configuration file:

```
/etc/cloudera-scm-agent/config.ini
```

On each Cloudera Manager Agent host:

1. Create symbolic links (symlink) for the certificate file:

```
$ ln -s /opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem
/opt/cloudera/security/pki/agent.cert.pem
```

2. Create symlink for the keystore file:

```
$ ln -s /opt/cloudera/security/pki/$(hostname -f)-agent.jks
/opt/cloudera/security/pki/agent.jks
```

Configuring the Cloudera Manager Agent Hosts for Certificate Authentication

The steps below assume that the Preliminary Tasks (obtaining and distributing the signed certificates) are complete.

Step 7: Export the Private Key to a File

Repeat these steps each Cloudera Manager Agent host system.

On each Cloudera Manager Agent host, use the `keytool` utility to export the private key and certificate to a PKCS12 file, which can then be split up into individual key and certificate files using the `openssl` command:

1. Create a PKCS12 version of the Java keystore:

```
$ keytool -importkeystore -srckeystore /opt/cloudera/security/pki/$(hostname -f)-agent.jks
 \
-srcstorepass password -srckeypass password \
-destkeystore /opt/cloudera/security/pki/$(hostname -f)-agent.p12 \
-deststoretype PKCS12 -srcalias $(hostname -f)-agent -deststorepass \
password -destkeypass password
```

2. Use the `openssl` command to export the private key into its own file:

```
$ openssl pkcs12 -in /opt/cloudera/security/pki/$(hostname -f)-agent.p12 \
-passin pass:password -nocerts -out \
/opt/cloudera/security/pki/$(hostname -f)-agent.key -passout pass:password
```

3. Create a symbolic link for the `.key` filename:

```
$ ln -s /opt/cloudera/security/pki/$(hostname -f)-agent.key
/opt/cloudera/security/pki/agent.key
```

With the symlink pointing to `agent.key` on each Cloudera Manager Agent host in the cluster, the same `/etc/cloudera-scm-agent/config.ini` file can also be used for all agent hosts rather than maintaining individual files. (The configuration changes are made in Step 8, below.)

Step 8: Create a Password File

The Cloudera Manager agent obtains the password from a text file, not from a command line parameter or environment variable. The password file allows you to use file permissions to protect the password. For example, run the following commands on each Cloudera Manager Agent host, or run them on one host and copy the file to the other hosts:

```
$ echo "password" > /etc/cloudera-scm-agent/agentkey.pw
$ sudo chown root:root /etc/cloudera-scm-agent/agentkey.pw
$ sudo chmod 440 /etc/cloudera-scm-agent/agentkey.pw
```

Replace *password* with the password you created in Step 7: Export the Private Key to a File on page 196.

Step 9: Configure the Agent to Use Private Keys and Certificates

Edit the configuration file on one of the Cloudera Manager Agent hosts in the cluster, as follows:

1. Use a text editor to open the configuration file, located in this path:

```
/etc/cloudera-scm-agent/config.ini
```

2. In the [Security] section in the file (just above the [Hadoop] section, find the PEM certificates section details, specifically:

a. `client_key_file`, which identifies the path to the private key file.
b. `client_keypw_file`, which identifies the path to the private key password file.
c. `client_cert_file`, which identifies the path to the client certificate file.

3. Remove the hash mark (#) at the start of the line and enter the values for your system, as shown below (in bold) using the settings created so far:

```
[security]
...
# PEM file containing client private key
client_key_file=/opt/cloudera/security/pki/agent.key
...
# If client_keypw_cmd isn't specified, instead a text file contining
# the client private key password can be used.
client_keypw_file=/etc/cloudera-scm-agent/agentkey.pw

# PEM file containing client certificate.
client_cert_file=/opt/cloudera/security/pki/agent.cert.pem
...
```

Copy the file to all other cluster hosts. If you have modified properties such as `listening_hostname` or `listening_ip address` in `config.ini`, you must edit the file individually on each host.

Step 10: Enable Agent Certificate Authentication

1. Log in to the Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Click the **Security** category.
4. Configure the following TLS settings:

Setting	Description
Use TLS Authentication of Agents to Server	Click the box to enable TLS authentication for Cloudera Manager Agent hosts. By default, the box is unchecked.
Cloudera Manager TLS/SSL Certificate Trust Store File	Enter the full filesystem path to the `jssecacerts` file (created in Step 2: Create the Java Truststore on page 187) on the Cloudera Manager Server host. For example: `/usr/java/jdk1.7.0_67-cloudera/jre/lib/security/jssecacerts`
Cloudera Manager TLS/SSL Certificate Trust Store Password	Specify the password for the `jssecacerts` truststore.

5. Click **Save Changes** to save the settings.

Step 11: Restart Cloudera Manager Server and Agents

1. On the Cloudera Manager server host, restart the Cloudera Manager server:

```
$ sudo service cloudera-scm-server restart
```

2. On every agent host, restart the Cloudera Manager agent:

```
$ sudo service cloudera-scm-agent restart
```

Step 12: Verify Cloudera Manager Server and Agent Communications

In the Cloudera Manager Admin Console, go to **Hosts** > **All Hosts**. If you see successful heartbeats reported in the **Last Heartbeat** column after restarting the agents and server, TLS certificate authentication is working properly. If not, check the agent log (`/var/log/cloudera-scm-agent/cloudera-scm-agent.log`) for errors. See TLS Level 2, Step 4 for details about accessing the log.

Configuring TLS/SSL Encryption for CDH Services

In addition to configuring Cloudera Manager cluster to use TLS/SSL (as detailed, starting with Configuring Cloudera Manager Clusters for TLS/SSL on page 183), the various CDH services running on the cluster should also be configured to use TLS/SSL. The process of configuring TLS/SSL varies by component, so follow the steps below as needed for your system. Before trying to configure TLS/SSL, however, be sure your cluster meets prerequisites.

In general, all the roles on any given node in the cluster can use the same certificates, assuming the certificates are in the appropriate format (JKS, PEM) and that the configuration properly points to the location. If you follow the steps in How to Configure TLS Encryption for Cloudera Manager to create your CSRs and use the symbolic link for the path to the certificates, you will be setting up the certificates in the cluster for optimal reuse.

> **Note:** TLS/SSL for Hadoop core services—HDFS, MapReduce, and YARN—must be enabled as a group. TLS/SSL for other components such as HBase, Hue, and Oozie can be enabled independently.

Not all components support TLS/SSL, nor do all external engines support TLS/SSL. Unless explicitly listed in this guide, the component you want to configure may not currently support TLS/SSL. For example, Sqoop does not currently support TLS/SSL to Oracle, MySQL, or other databases.

Prerequisites

Cloudera recommends that the cluster and all services use Kerberos for authentication. If you enable TLS/SSL for a cluster that has not been configured to use Kerberos, a warning displays. You should integrate the cluster with your Kerberos deployment before proceeding.

The steps below require the cluster to have been configured to a minimum of TLS Level 2, to ensure that Cloudera Manager Server certificate and Cloudera Manager Agent certificates are properly configured and already in place. In addition, you should have the certificates and keys needed by the specific CDH server ready.

If the cluster meets these requirements, you can configure the specific CDH service to use TLS/SSL, as detailed in this section.

Configuring TLS/SSL for HDFS, YARN and MapReduce

Required Role: Configurator, Cluster Administrator, or Full Administrator

TLS/SSL for the core Hadoop services—HDFS, MapReduce, and YARN—must be enabled as a group. Because most clusters run either MapReduce or YARN, not both, you will typically enable HDFS and YARN, or HDFS and MapReduce. Enabling TLS/SSL on HDFS is required before it can be enabled on either MapReduce or YARN.

> **Note:** If you enable TLS/SSL for HDFS, you must also enable it for MapReduce or YARN.

The steps below include enabling Kerberos authentication for HTTP Web-Consoles. Enabling TLS/SSL for the core Hadoop services on a cluster without enabling authentication displays a warning.

Before You Begin

- Before enabling TLS/SSL, keystores containing certificates bound to the appropriate domain names will need to be accessible on all hosts on which at least one HDFS, MapReduce, or YARN daemon role is running.
- Since HDFS, MapReduce, and YARN daemons act as TLS/SSL clients as well as TLS/SSL servers, they must have access to truststores. In many cases, the most practical approach is to deploy truststores to all hosts in the cluster, as it may not be desirable to determine in advance the set of hosts on which clients will run.
- Keystores for HDFS, MapReduce and YARN must be owned by the hadoop group, and have permissions `0440` (that is, readable by owner and group). Truststores must have permissions `0444` (that is, readable by all)
- Cloudera Manager supports TLS/SSL configuration for HDFS, MapReduce and YARN at the service level. For each of these services, you must specify absolute paths to the keystore and truststore files. These settings apply to all

hosts on which daemon roles of the service in question run. Therefore, the paths you choose must be valid on all hosts.

An implication of this is that the keystore file names for a given service must be the same on all hosts. If, for example, you have obtained separate certificates for HDFS daemons on hosts `node1.example.com` and `node2.example.com`, you might have chosen to store these certificates in files called `hdfs-node1.keystore` and `hdfs-node2.keystore` (respectively). When deploying these keystores, you must give them both the same name on the target host — for example, `hdfs.keystore`.

- Multiple daemons running on a host can share a certificate. For example, in case there is a DataNode and an Oozie server running on the same host, they can use the same certificate.

Configuring TLS/SSL for HDFS

1. Go to the **HDFS** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HDFS (Service-Wide)**.
4. Select **Category** > **Security**.
5. In the Search field, type **TLS/SSL** to show the TLS/SSL properties (found under the **Service-Wide** > **Security** category).
6. Edit the following properties according to your cluster configuration:

Property	Description
Hadoop TLS/SSL Server Keystore File Location	Path to the keystore file containing the server certificate and private key.
Hadoop TLS/SSL Server Keystore File Password	Password for the server keystore file.
Hadoop TLS/SSL Server Keystore Key Password	Password that protects the private key contained in the server keystore.

7. If you are not using the default truststore, configure TLS/SSL client truststore properties:

> **Important:** The HDFS properties below define a cluster-wide default truststore that can be overridden by YARN and MapReduce (see the **Configuring TLS/SSL for YARN and MapReduce** section below).

Property	Description
Cluster-Wide Default TLS/SSL Client Truststore Location	Path to the client truststore file. This truststore contains certificates of trusted servers, or of Certificate Authorities trusted to identify servers.
Cluster-Wide Default TLS/SSL Client Truststore Password	Password for the client truststore file.

8. Cloudera recommends you enable Web UI authentication for the HDFS service.

   Enter **web consoles** in the Search field to bring up the **Enable Authentication for HTTP Web-Consoles** property (found under the **Service-Wide**>**Security** category). Check the property to enable web UI authentication.

Enable Authentication for HTTP Web-Consoles	Enables authentication for Hadoop HTTP web-consoles for all roles of this service. **Note:** This is effective only if security is enabled for the HDFS service.

9. Click **Save Changes**.

10. Follow the procedure described in the following **Configuring TLS/SSL for YARN and MapReduce** section, at the end of which you will be instructed to restart all the affected services (HDFS, MapReduce and YARN).

Configuring TLS/SSL for YARN or MapReduce

Perform the following steps to configure TLS/SSL for the YARN or MapReduce services:

1. Go to the **YARN** or **MapReduce** service.
2. Click the **Configuration** tab.
3. Select **Scope** > ***service name*** **(Service-Wide)**.
4. Select **Category** > **Security**.
5. Locate the **<property name>** property or search for it by typing its name in the Search box.
6. In the Search field, type **TLS/SSL** to show the TLS/SSL properties (found under the **Service-Wide** > **Security** category).
7. Edit the following properties according to your cluster configuration:

Property	Description
Hadoop TLS/SSL Server Keystore File Location	Path to the keystore file containing the server certificate and private key.
Hadoop TLS/SSL Server Keystore File Password	Password for the server keystore file.
Hadoop TLS/SSL Server Keystore Key Password	Password that protects the private key contained in the server keystore.

8. Configure the following TLS/SSL client truststore properties for MRv1 or YARN only if you want to override the cluster-wide defaults set by the HDFS properties configured above.

Property	Description
TLS/SSL Client Truststore File Location	Path to the client truststore file. This truststore contains certificates of trusted servers, or of Certificate Authorities trusted to identify servers.
TLS/SSL Client Truststore File Password	Password for the client truststore file.

9. Cloudera recommends you enable Web UI authentication for the service in question.

   Enter **web consoles** in the Search field to bring up the **Enable Authentication for HTTP Web-Consoles** property (found under the **Service-Wide>Security** category). Check the property to enable web UI authentication.

Enable Authentication for HTTP Web-Consoles	Enables authentication for Hadoop HTTP web-consoles for all roles of this service. **Note:** This is effective only if security is enabled for the HDFS service.

10. Click **Save Changes** to commit the changes.
11. Go to the **HDFS** service
12. Click the **Configuration** tab.
13. Type **Hadoop SSL Enabled** in the Search box.
14. Select the **Hadoop SSL Enabled** property to enable SSL communication for HDFS, MapReduce, and YARN.

Property	Description
Hadoop TLS/SSL Enabled	Enable TLS/SSL encryption for HDFS, MapReduce, and YARN web UIs, as well as encrypted shuffle for MapReduce and YARN.

15. Click **Save Changes** to commit the changes.
16. Restart all affected services (HDFS, MapReduce and YARN), as well as their dependent services.

Configuring TLS/SSL for HBase

Required Role: Configurator, Cluster Administrator, or Full Administrator

Before You Begin

- Before enabling TLS/SSL, ensure that keystores containing certificates bound to the appropriate domain names will need to be accessible on all hosts on which at least one HBase daemon role is running.
- Keystores for HBase must be owned by the `hbase` group, and have permissions `0440` (that is, readable by owner and group).
- You must specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the HBase service run. Therefore, the paths you choose must be valid on all hosts.
- Cloudera Manager supports the TLS/SSL configuration for HBase at the service level. Ensure you specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the service in question run. Therefore, the paths you choose must be valid on all hosts.

  An implication of this is that the keystore file names for a given service must be the same on all hosts. If, for example, you have obtained separate certificates for HBase daemons on hosts `node1.example.com` and `node2.example.com`, you might have chosen to store these certificates in files called `hbase-node1.keystore` and `hbase-node2.keystore` (respectively). When deploying these keystores, you must give them both the same name on the target host — for example, `hbase.keystore`.

Configuring TLS/SSL for HBase Web UIs

The steps for configuring and enabling TLS/SSL for HBase are similar to those for HDFS, YARN and MapReduce:

1. Go to the **HBase service**.
2. Click the **Configuration** tab.
3. Select **Scope** > **HBASE (Service-Wide)**.
4. Select **Category** > **Security**.
5. In the Search field, type **TLS/SSL** to show the HBase TLS/SSL properties.
6. Edit the following TLS/SSL properties according to your cluster configuration:

Table 10: HBase TLS/SSL Properties

Property	Description
TLS/SSL Server Keystore File Location	Path to the keystore file containing the server certificate and private key used for encrypted web UIs.
TLS/SSL Server Keystore File Password	Password for the server keystore file used for encrypted web UIs.
TLS/SSL Server Keystore Key Password	Password that protects the private key contained in the server keystore used for encrypted web UIs.

7. Check the **Web UI TLS/SSL Encryption Enabled** property.

Web UI TLS/SSL Encryption Enabled	Enable TLS/SSL encryption for the HBase Master, RegionServer, Thrift Server, and REST Server web UIs.

8. Click **Save Changes**.
9. Restart the HBase service.

Configuring TLS/SSL for HBase REST Server

1. Go to the **HBase service**
2. Click the **Configuration** tab.
3. Select **Scope** > **HBase REST Server**.
4. Select **Category** > **Security**.
5. In the Search field, type **TLS/SSL REST** to show the HBase REST TLS/SSL properties.
6. Edit the following TLS/SSL properties according to your cluster configuration:

Property	Description
Enable TLS/SSL for HBase REST Server	Encrypt communication between clients and HBase REST Server using Transport Layer Security (TLS).
HBase REST Server TLS/SSL Server JKS Keystore File Location	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when HBase REST Server is acting as a TLS/SSL server. The keystore must be in JKS format.file.
HBase REST Server TLS/SSL Server JKS Keystore File Password	The password for the HBase REST Server JKS keystore file.
HBase REST Server TLS/SSL Server JKS Keystore Key Password	The password that protects the private key contained in the JKS keystore used when HBase REST Server is acting as a TLS/SSL server.

7. Click **Save Changes**.
8. Restart the HBase service.

Configuring TLS/SSL for HBase Thrift Server

1. Go to the **HBase service**
2. Click the **Configuration** tab.
3. Select **Scope** > **HBase Thrift Server**.
4. Select **Category** > **Security**.
5. In the Search field, type **TLS/SSL Thrift** to show the HBase Thrift TLS/SSL properties.
6. Edit the following TLS/SSL properties according to your cluster configuration:

Property	Description
Enable TLS/SSL for HBase Thrift Server over HTTP	Encrypt communication between clients and HBase Thrift Server over HTTP using Transport Layer Security (TLS).
HBase Thrift Server over HTTP TLS/SSL Server JKS Keystore File Location	Path to the TLS/SSL keystore file (in JKS format) with the TLS/SSL server certificate and private key. Used when HBase Thrift Server over HTTP acts as a TLS/SSL server.
HBase Thrift Server over HTTP TLS/SSL Server JKS Keystore File Password	Password for the HBase Thrift Server JKS keystore file.
HBase Thrift Server over HTTP TLS/SSL Server JKS Keystore Key Password	Password that protects the private key contained in the JKS keystore used when HBase Thrift Server over HTTP acts as a TLS/SSL server.

7. Click **Save Changes**.
8. Restart the HBase service.

Configuring TLS/SSL for Flume Thrift Source and Sink

This topic describes how to enable TLS/SSL communication between Flume's Thrift source and sink.

The following tables list the properties that must be configured to enable TLS/SSL communication between Flume's Thrift source and sink instances.

Table 11: Thrift Source TLS/SSL Properties

Property	Description
`ssl`	Set to `true` to enable TLS/SSL encryption.
`keystore`	Path to a Java keystore file. Required for TLS/SSL.
`keystore-password`	Password for the Java keystore. Required for TLS/SSL.
`keystore-type`	The type of the Java keystore. This can be JKS or PKCS12.

Table 12: Thrift Sink TLS/SSL Properties

Property	Description
`ssl`	Set to `true` to enable TLS/SSL for this ThriftSink. When configuring TLS/SSL, you can optionally set the following `truststore`, `truststore-password` and `truststore-type` properties. If a custom truststore is not specified, Flume will use the default Java JSSE truststore (typically `jssecacerts` or `cacerts` in the Oracle JRE) to verify the remote Thrift Source's TLS/SSL credentials.
`truststore`	(Optional) The path to a custom Java truststore file.
`truststore-password`	(Optional) The password for the specified truststore.
`truststore-type`	(Optional) The type of the Java truststore. This can be JKS or any other supported Java truststore type.

Make sure you are configuring TLS/SSL for ***each*** Thrift source and sink instance. For example, to the existing `flume.conf` file, for agent `a1`, source `r1`, and sink `k1`, you would add the following properties:

```
# TLS/SSL properties for Thrift source s1
a1.sources.r1.ssl=true
a1.sources.r1.keystore=<path/to/keystore>
a1.sources.r1.keystore-password=<keystore password>
a1.sources.r1.keystore-type=<keystore type>

# TLS/SSL properties for Thrift sink k1
a1.sinks.k1.ssl=true
a1.sinks.k1.truststore=<path/to/truststore>
a1.sinks.k1.truststore-password=<truststore password>
a1.sinks.k1.truststore-type=<truststore type>
```

Configure these sets of properties for more instances of the Thrift source and sink as required. You can use either Cloudera Manager or the command line to edit the `flume.conf` file.

Using Cloudera Manager

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

1. Open the Cloudera Manager Admin Console and go to the **Flume** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Agent**.
4. Select **Category** > **Main**.
5. Edit the **Configuration File** property and add the Thrift source and sink properties for each Thrift source and sink instance as described above to the configuration file.
6. Click **Save Changes** to commit the changes.
7. Restart the Flume service.

Using the Command Line

Go to the `/etc/flume-ng/conf/flume.conf` file and add the Thrift source and sink properties for each Thrift source and sink instance as described above.

Configuring Encrypted Communication Between HiveServer2 and Client Drivers

Starting with CDH 5.5, encryption for HiveServer2 clients has been decoupled from the authentication mechanism. This means you can use either SASL QOP or TLS/SSL to encrypt traffic between HiveServer2 and its clients, irrespective of whether Kerberos is being used for authentication. Previously, the JDBC client drivers only supported SASL QOP encryption on Kerberos-authenticated connections.

SASL QOP encryption is better suited for encrypting RPC communication and may result in performance issues when dealing with large amounts of data. Move to using TLS/SSL encryption to avoid such issues.

This topic describes how to set up encrypted communication between HiveServer2 and its JDBC/ODBC client drivers.

Configuring Encrypted Client/Server Communication Using TLS/SSL

You can use either the Cloudera Manager or the command-line instructions described below to enable TLS/SSL encryption for JDBC/ODBC client connections to HiveServer2. For background information on setting up TLS/SSL truststores and keystores, see Data in Transit Encryption (TLS/SSL) on page 179.

> **Note:** Cloudera Manager and CDH components support either TLS 1.0, TLS 1.1, or TLS 1.2, but not SSL 3.0. References to SSL continue only because of its widespread use in technical jargon.

Using Cloudera Manager

The steps for configuring and enabling TLS/SSL for Hive are as follows:

1. Open the Cloudera Manager Admin Console and go to the Hive service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Hive (Service-Wide)**.
4. Select **Category** > **Security**.
5. In the Search field, type **TLS/SSL** to show the Hive properties.
6. Edit the following properties according to your cluster configuration.

Table 13: Hive TLS/SSL Properties

Property	Description
Enable TLS/SSL for HiveServer2	Enable support for encrypted client-server communication using Transport Layer Security (TLS) for HiveServer2 connections.
HiveServer2 TLS/SSL Server JKS Keystore File Location	Path to the TLS keystore.
HiveServer2 TLS/SSL Server JKS Keystore File Password	Password for the TLS keystore.

7. Click **Save Changes** to commit the changes.
8. Restart the Hive service.

Using the Command Line

- To enable TLS/SSL, add the following configuration parameters to `hive-site.xml` :

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

```
</property>

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

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

- The `keystore` must contain the server's certificate.
- The JDBC client must add the following properties in the connection URL when connecting to a HiveServer2 using TLS/SSL:

```
;ssl=true[;sslTrustStore=<Trust-Store-Path>;trustStorePassword=<Trust-Store-password>]
```

- Make sure one of the following is true:
  - *Either:* `sslTrustStore` points to the truststore file containing the server's certificate; for example:

```
jdbc:hive2://localhost:10000/default;ssl=true;\
sslTrustStore=/home/usr1/ssl/trust_store.jks;trustStorePassword=xyz
```

  - *or:* the Trust Store arguments are set using the Java system properties `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword`; for example:

```
java -Djavax.net.ssl.trustStore=/home/usr1/ssl/trust_store.jks
-Djavax.net.ssl.trustStorePassword=xyz \
 MyClass jdbc:hive2://localhost:10000/default;ssl=true
```

For more information on using self-signed certificates and the Trust Store, see the Oracle Java SE keytool page.

Configuring Encrypted Client/Server Communication Using SASL QOP

Traffic between the Hive JDBC or ODBC drivers and HiveServer2 can be encrypted using plain SASL QOP encryption which allows you to preserve data integrity (using checksums to validate message integrity) and confidentiality (by encrypting messages). This can be enabled by setting the `hive.server2.thrift.sasl.qop` property in `hive-site.xml`. For example,

```
<property>
<name>hive.server2.thrift.sasl.qop</name>
<value>auth-conf</value>
<description>Sasl QOP value; one of 'auth', 'auth-int' and 'auth-conf'</description>
</property>
```

Valid settings for the `value` field are:

- `auth`: Authentication only (default)
- `auth-int`: Authentication with integrity protection
- `auth-conf`: Authentication with confidentiality protection

The parameter value that you specify above in the HiveServer2 configuration, should match that specified in the Beeline client connection JDBC URL. For example:

```
!connect jdbc:hive2://ip-10-5-15-197.us-west-2.compute.internal:10000/default;  \
principal=hive/_HOST@US-WEST-2.COMPUTE.INTERNAL;sasl.qop=auth-conf
```

Configuring TLS/SSL for Hue

Hue as a TLS/SSL Client

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

Hue acts as a TLS/SSL client when communicating with other services such as core Hadoop, HBase, Oozie and Amazon S3. This means Hue must authenticate HDFS, MapReduce, YARN daemons, the HBase Thrift server, and so on. To do so, Hue needs the certificate chains of their hosts in its truststore.

The Hue truststore is a single PEM file that contains the CA root, and all intermediate certificates, to authenticate the certificate installed on each TLS/SSL-enabled server. These servers host the various services with which Hue communicates.

> **Note:** A certificate is specific to a host. It is signed by a certificate authority (CA) and tells the requesting client (Hue) that "this host" is the same one as represented by the host public key. Hue uses chain of signing authorities in its truststore to validate the CA that signed the host certificate.

Creating a Hue Truststore File in PEM Format

Server certificates are stored in JKS format and must be converted to PEM. To create the Hue truststore, extract each certificate from its keystore with `keytool`, convert to PEM format with `openssl`, and add to the truststore.

1. Extract the certificate from the keystore of each TLS/SSL-enabled server with which Hue communicates.

   For example, `hadoop-server.keystore` contains server certificate, `foo-1.example.com`, and password, `example123`.

```
keytool -exportcert -keystore hadoop-server.keystore -alias foo-1.example.com -storepass
 example123 -file foo-1.cert
```

2. Convert each certificate into a PEM file.

```
openssl x509 -inform der -in foo-1.cert > foo-1.pem
```

3. Concatenate all the PEM certificates into one PEM file.

```
cat foo-1.pem foo-2.pem foo-n.pem ... > hue_trustore.pem
```

> **Note:** Ensure the final PEM truststore is deployed in a location that is accessible by the Hue service.

Configuring Hue as a TLS/SSL Client with Cloudera Manager

1. Go to the **Hue** service and click the **Configuration** tab.
2. Filter by **Scope** > **Hue Server** and **Category** > **Security**.
3. Find the property, `Hue TLS/SSL Server CA Certificate (PEM Format)`, or `ssl_cacerts`.
4. Enter the path to <hue_truststore>.pem on the host running the Hue web server.
5. Click **Save Changes**.
6. Select **Actions** > **Restart** to restart the Hue service.

Configuring Hue as a TLS/SSL Client at the Command Line

For unmanaged deployments only, manually set `ssl_cacerts` in `hue.ini` to the path of the <hue_truststore>.pem file:

```
[desktop]
# Path to default Certificate Authority certificates.
ssl_cacerts=/etc/hue/<hue_truststore>.pem
```

Hue as a TLS/SSL Server

Hue and other Python-based services expect certificates and keys to be stored in PEM format. You can manage such services with the openssl tool. To configure Hue to use HTTPS, generate a private key and certificate as described in Obtain and Deploy Server Certificate on page 185 and reuse a host's existing Java keystore by converting it to the PEM format. See Converting JKS Key and Certificate to PEM on page 465.

Enabling TLS/SSL for the Hue Server with Cloudera Manager

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

1. Go to the **Hue service** and click **Configuration**.
2. Filter by **Scope** > **Hue Server** and **Category** > **Security**.
3. Edit the following **TLS/SSL** properties according to your cluster configuration.

Property	Description
Enable TLS/SSL for Hue	Encrypt communication between clients and Hue with TLS/SSL.
Hue TLS/SSL Server Certificate File (PEM Format) `ssl_certificate`	Path to TLS/SSL certificate on host running Hue web server.
Hue TLS/SSL Server Private Key File (PEM Format `ssl_private_key`	Path to TLS/SSL private key on host running Hue web server.
Hue TLS/SSL Private Key Password `ssl_password`	Password for private key in Hue TLS/SSL Server Certificate and Private Key file.

You can also store `ssl_password` more securely in a script and set this parameter instead:

```
ssl_password_script=<your_hue_passwords_script.sh>
```

For more, see Storing Hue Passwords in a Script on page 209.

To apply this configuration property to other role groups as needed, edit the value for the appropriate role group. See Modifying Configuration Properties Using Cloudera Manager.

4. Click **Save Changes**.
5. Select **Actions** > **Restart** to restart the Hue service.

For more details on configuring Hue with TLS/SSL, see this blog post.

Enabling TLS/SSL for the Hue Server at the Command Line

1. Enable secure session cookies in `hue.ini` under `[desktop]>[[session]]`.

```
[desktop]
  [[session]]
  secure=true
```

2. Edit the following properties in `hue.ini` under `[desktop]`.

```
[desktop]
 ssl_certificate=/path/to/server.cert
 ssl_private_key=/path/to/server.key
 ssl_password=<private_key_password>
```

You can store `ssl_password` more securely in a script and set this parameter instead:

```
ssl_password_script=<your_hue_passwords_script.sh>
```

For more, see Storing Hue Passwords in a Script on page 209.

Enabling Hue TLS/SSL Communication with HiveServer2

In CDH 5.5.x and higher, HiveServer2 is enabled for TLS/SSL communication by default.

By providing a CA certificate, private key, and public certificate, Hue can communicate with HiveServer2 over TLS/SSL. You can now configure the following properties in the `[beeswax]` section under `[[ssl]]` in the Hue configuration file, `hue.ini`.

`enabled`	Choose to enable/disable TLS/SSL communication for this server. Default: `false`
`cacerts`	Path to Certificate Authority certificates. Default: `/etc/hue/cacerts.pem`
`validate`	Choose whether Hue should validate certificates received from the server. Default: `true`

Related Information

- Configuring Encrypted Communication Between HiveServer2 and Client Drivers on page 204

Enabling Hue TLS/SSL Communication with Impala

In CDH 5.5.x and higher, Impala is enabled for TLS/SSL communication by default.

By providing a CA certificate, private key, and public certificate, Hue can communicate with Impala over TLS/SSL. You can configure the following properties in the `[impala]` section under `[[ssl]]` in the Hue configuration file, `hue.ini`.

`enabled`	Choose to enable/disable TLS/SSL communication for this server. Default: `false`
`cacerts`	Path to Certificate Authority certificates. Default: `/etc/hue/cacerts.pem`
`validate`	Choose whether Hue should validate certificates received from the server. Default: `true`

Securing Database Connections using TLS/SSL

Connections vary depending on the database. Hue uses different clients to communicate with each database internally. Client specific options, such as secure connectivity, can be passed through the interface.

For example, for MySQL you can enable TLS/SSL communication by specifying the `options` configuration property under the `desktop>[[database]]` section in `hue.ini`. Here we identify the Certificate Authority (CA) certificate:

```
[desktop]
  [[databases]]
    …
    options={"ssl":{"ca":"/tmp/ca-cert.pem"}}
```

You can also identify public and private keys, for example:

```
options='{"ssl": {"ca": "/tmp/newcerts2/ca.pem", "key": "/tmp/newcerts2/client-key.pem",
 "cert": "/tmp/newcerts2/client-cert.pem"}}'
```

Storing Hue Passwords in a Script

In CDH 5.4, Hue added the ability to store passwords in a secure script and pull passwords from `stdout`. On startup, Hue runs one or more passwords scripts and grabs each password from `stdout`.

In `hue_ini`, add the suffix, `_script`, to any password property and set it equal to the script name. In Cloudera Manager, set these properties in the configuration field, **Hue Service Advanced Configuration Snippet (Safety Valve) for hue_safety_valve.ini**. For example:

```
[desktop]
ldap_username=hueservice
ldap_password_script="/var/lib/hue/<your_hue_passwords_script.sh> ldap_password"
ssl_password_script="/var/lib/hue/<your_hue_passwords_script.sh> ssl_password"

[[ldap]]
bind_password_script="/var/lib/hue/<your_hue_passwords_script.sh> bind_password"

[[database]]
password_script="/var/lib/hue/<your_hue_passwords_script.sh> database"
```

Store the script in a directory that only the hue user can read, write, and execute. You can have one script per password or one script with parameters for all passwords. Here is an example of a script with parameters for multiple passwords:

```
#!/bin/bash

SERVICE=$1

if [[ ${SERVICE} == "ldap_password" ]]
then
   echo "password"
fi

if [[ ${SERVICE} == "ssl_password" ]]
then
   echo "password"
fi

if [[ ${SERVICE} == "bind_password" ]]
then
   echo "Password1"
fi

if [[ ${SERVICE} == "database_password" ]]
then
   echo "password"
fi
```

Note: The bind password parameter was added in CDH 5.4.6.

Configuring TLS/SSL for Impala

Impala supports TLS/SSL network encryption, between Impala and client programs, and between the Impala-related daemons running on different nodes in the cluster. This feature is important when you also use other features such as Kerberos authentication or Sentry authorization, where credentials are being transmitted back and forth.

Important:

- You can use either Cloudera Manager or the following command-line instructions to complete this configuration.
- This information applies specifically to the version of Impala shown in the HTML page header or on the PDF title page. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

Using Cloudera Manager

To configure Impala to listen for Beeswax and HiveServer2 requests on TLS/SSL-secured ports:

1. Open the Cloudera Manager Admin Console and go to the **Impala** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Impala (Service-Wide)**.
4. Select **Category** > **Security**.
5. Edit the following properties:

Table 14: Impala SSL Properties

Property	Description
Enable TLS/SSL for Impala Client Services	Encrypt communication between clients (like ODBC, JDBC, and the Impala shell) and the Impala daemon using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
SSL/TLS Certificate for Clients	Local path to the X509 certificate that identifies the Impala daemon to clients during TLS/SSL connections. This file must be in PEM format.
SSL/TLS Private Key for Clients	Local path to the private key that matches the certificate specified in the Certificate for Clients. This file must be in PEM format.
SSL/TLS Private Key Password for Clients	A shell command for Impala to run on startup to retrieve the password for a password-protected private key file. The output of the command is truncated to a maximum of 1024 bytes, and any trailing whitespace (such as spaces or newline characters) is trimmed. If the command exits with an error, Impala does not start. If the password is incorrect, clients cannot connect to the server regardless of whether the public key is correct.
SSL/TLS CA Certificate	Must be specified for TLS/SSL encryption to be enabled for communication between internal Impala components.
SSL/TLS Certificate for *Impala component* Webserver	There are three of these configuration settings, one each for “Impala Daemon”, “Catalog Server”, and “Statestore”. Each of these Impala components has its own internal web server that powers the associated web UI with diagnostic information. The configuration setting represents the local path to the X509 certificate that identifies the web server to clients during TLS/SSL connections. This file must be in PEM format.

6. Click **Save Changes** to commit the changes.
7. Restart the Impala service.

For information on configuring TLS/SSL communication with the `impala-shell` interpreter, see Configuring TLS/SSL Communication for the Impala Shell on page 211.

Using the Command Line

To enable SSL for when client applications connect to Impala, add both of the following flags to the `impalad` startup options:

- `--ssl_server_certificate`: the full path to the server certificate, on the local filesystem.
- `--ssl_private_key`: the full path to the server private key, on the local filesystem.

In and higher, Impala can also use SSL for its own internal communication between the `impalad`, `statestored`, and `catalogd` daemons. To enable this additional SSL encryption, set the `--ssl_server_certificate` and `--ssl_private_key` flags in the startup options for `impalad`, `catalogd`, and `statestored`, and also add the `--ssl_client_ca_certificate` flag for all three of those daemons.

> **Warning:** Prior to CDH 5.5.2 / Impala 2.3.2, you could enable Kerberos authentication between Impala internal components, or SSL encryption between Impala internal components, but not both at the same time. This restriction has now been lifted. See IMPALA-2598 to see the maintenance releases for different levels of CDH where the fix has been published.

If either of these flags are set, both must be set. In that case, Impala starts listening for Beeswax and HiveServer2 requests on SSL-secured ports only. (The port numbers stay the same; see Ports Used by Impala for details.)

Since Impala uses passphrase-less certificates in PEM format, you can reuse a host's existing Java keystore by converting it to the PEM format. For instructions, see Converting JKS Key and Certificate to PEM on page 465.

Configuring TLS/SSL Communication for the Impala Shell

Typically, a client program has corresponding configuration properties in Cloudera Manager to verify that it is connecting to the right server. For example, with SSL enabled for Impala, you use the following options when starting the `impala-shell` interpreter:

- `--ssl`: enables TLS/SSL for `impala-shell`.
- `--ca_cert`: the local pathname pointing to the third-party CA certificate, or to a copy of the server certificate for self-signed server certificates.

If `--ca_cert` is not set, `impala-shell` enables TLS/SSL, but does not validate the server certificate. This is useful for connecting to a known-good Impala that is only running over TLS/SSL, when a copy of the certificate is not available (such as when debugging customer installations).

Using TLS/SSL with Business Intelligence Tools

You can use Kerberos authentication, TLS/SSL encryption, or both to secure connections from JDBC and ODBC applications to Impala. See Configuring Impala to Work with JDBC and Configuring Impala to Work with ODBC for details.

Prior to CDH 5.7 / Impala 2.5, the Hive JDBC driver did not support connections that use both Kerberos authentication and SSL encryption. If your cluster is running an older release that has this restriction, to use both of these security features with Impala through a JDBC application, use the Cloudera JDBC Connector as the JDBC driver.

Configuring TLS/SSL for Oozie

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

Before You Begin

- Keystores for Oozie must be readable by the `oozie` user. This could be a copy of the Hadoop services' keystore with permissions `0440` and owned by the `oozie` group.

- Truststores must have permissions `0444` (that is, readable by all).
- Specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the Oozie service run. Therefore, the paths you choose must be valid on all hosts.
- In case there is a DataNode and an Oozie server running on the same host, they can use the same certificate.

For more information on obtaining signed certificates and creating keystores, see Data in Transit Encryption (TLS/SSL) on page 179. You can also view the upstream documentation located here.

Important:

- This configuration process can be completed using either Cloudera Manager or the command-line instructions.
- This information applies specifically to CDH 5.13.x. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

Using Cloudera Manager

The steps for configuring and enabling Hadoop TLS/SSL for Oozie are as follows:

1. Open the Cloudera Manager Admin Console and go to the **Oozie service**.
2. Click the **Configuration** tab.
3. Select **Scope** > **All**.
4. Select **Category** > **All**.
5. In the Search field, type **TLS/SSL** to show the Oozie TLS/SSL properties.
6. Edit the following TLS/SSL properties according to your cluster configuration.

Table 15: Oozie TLS/SSL Properties

Property	Description
Enable TLS/SSL for Oozie	Check this field to enable TLS/SSL for Oozie.
Oozie TLS/SSL Server Keystore File Location	Location of the keystore file on the local file system.
Oozie TLS/SSL Server JKS Keystore File Password	Password for the keystore.

7. Click **Save Changes**.
8. Restart the Oozie service.

Using the Command Line

To configure the Oozie server to use TLS/SSL:

1. Stop Oozie by running

```
sudo /sbin/service oozie stop
```

2. To enable TLS/SSL, set the MapReduce version that the Oozie server should work with using the `alternatives` command.

Note: The `alternatives` command is only available on RHEL systems. For SLES, Ubuntu and Debian systems, the command is `update-alternatives`.

For RHEL systems, to use YARN with TLS/SSL:

```
alternatives --set oozie-tomcat-conf /etc/oozie/tomcat-conf.https
```

For RHEL systems, to use MapReduce (MRv1) with TLS/SSL:

```
alternatives --set oozie-tomcat-conf /etc/oozie/tomcat-conf.https.mr1
```

> **Important:**
>
> The `OOZIE_HTTPS_KEYSTORE_PASS` variable must be the same as the password used when creating the keystore file. If you used a password other than `password`, you'll have to change the value of the `OOZIE_HTTPS_KEYSTORE_PASS` variable in this file.

3. Start Oozie by running

```
sudo /sbin/service oozie start
```

Connect to the Oozie Web UI using TLS/SSL (HTTPS)

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

Additional Considerations when Configuring TLS/SSL for Oozie HA

To allow clients to talk to Oozie servers (the target servers) through the load balancer using TLS/SSL, Configure the load balancer for TLS/SSL pass-through, which means the load balancer does not perform encryption/decryption but simply passes traffic from clients and servers to the appropriate target host. See documentation for your load balancer for details.

Configuring TLS/SSL for Solr

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

Before You Begin

- The Solr service must be running.
- Keystores for Solr must be readable by the `solr` user. This could be a copy of the Hadoop services' keystore with permissions `0440` and owned by the `solr` group.
- Truststores must have permissions `0444` (that is, readable by all).
- Specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the Solr service run. Therefore, the paths you choose must be valid on all hosts.
- In case there is a DataNode and a Solr server running on the same host, they can use the same certificate.

For more information on obtaining signed certificates and creating keystores, see Data in Transit Encryption (TLS/SSL) on page 179. You can also view the upstream documentation located here.

> **Important:**
>
> - This configuration process can be completed using either Cloudera Manager or the command-line instructions.
> - This information applies specifically to CDH 5.13.x. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

Additional Considerations when Configuring TLS/SSL for Solr HA

To allow clients to talk to Solr servers (the target servers) through the load balancer using TLS/SSL, Configure the load balancer for TLS/SSL pass-through, which means the load balancer does not perform encryption/decryption but simply passes traffic from clients and servers to the appropriate target host. See documentation for your load balancer for details.

Configuring TLS/SSL for Solr Using Cloudera Manager

The steps for configuring and enabling Hadoop TLS/SSL for Search are as follows:

1. Open the Cloudera Manager Admin Console and go to the **Solr service**.
2. Click the **Configuration** tab.
3. Select **Scope** > **All**.
4. Select **Category** > **All**.
5. In the Search field, type **TLS/SSL** to show the Solr TLS/SSL properties.
6. Edit the following properties according to your cluster configuration.

> **Note:** These values must be the same for all hosts running the Solr role.

Table 16: Solr TLS/SSL Properties

Property	Description
Enable TLS/SSL for Solr	Check this field to enable SSL for Solr.
Solr TLS/SSL Server Keystore File Location	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Solr is acting as a TLS/SSL server. The keystore must be in JKS format.
Solr TLS/SSL Server JKS Keystore File Password	Password for the Solr JKS keystore.
Solr TLS/SSL Certificate Trust Store File	Required in case of self-signed or internal CA signed certificates. The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that Solr might connect to. This is used when Solr is the client in a TLS/SSL connection. This truststore must contain the certificate(s) used to sign the service(s) being connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
Solr TLS/SSL Certificate Trust Store Password	The password for the Solr TLS/SSL Certificate Trust Store File. This password is not required to access the truststore: this field can be left blank. This password provides optional integrity checking of the file. The contents of truststores are certificates, and certificates are public information.

7. Click **Save Changes** to commit the changes.
8. Restart the service.

Additional Considerations When Using a Load Balancer TLS/SSL for Solr HA

To configure a load balancer:

1. Go to the Solr service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Solr**.
4. Enter the hostname and port number of the load balancer in the **Solr Load Balancer** property in the format `hostname:port number`.

> **Note:**
>
> When you set this property, Cloudera Manager regenerates the keytabs for Solr roles. The principal in these keytabs contains the load balancer hostname.
>
> If there are services that depends on this Solr service, such as Hue, those services use the load balancer to communicate with Solr.

5. Click **Save Changes** to commit the changes.
6. Restart Solr and any dependent services or restart the entire cluster for this configuration to take effect.

Configuring TLS/SSL for Solr Using the Command Line

To configure the Search to use TLS/SSL:

1. Use `solrctl` to modify the `urlScheme` setting to specify `https`. For example:

```
solrctl --zk myZKEnsemble:2181/solr cluster --set-property urlScheme https
```

2. Stop Solr by running

```
sudo service solr-server stop
```

3. Edit `/etc/default/solr` to include the following environment variable settings:

```
SOLR_SSL_ENABLED=true
SOLR_KEYSTORE_PATH=<absolute_path_to_keystore_file>
SOLR_KEYSTORE_PASSWORD=<keystore_password>

#Following required only in case of self-signed or internal CA signed certificates
SOLR_TRUSTSTORE_PATH=<absolute_path_to_truststore_file>
SOLR_TRUSTSTORE_PASSWORD=<truststore_password>
```

4. Start Solr by running

```
sudo service solr-server start
```

Configuring TLS/SSL for the Key-Value Store Indexer Using Cloudera Manager

The steps for configuring and enabling Hadoop TLS/SSL for the Keystore Indexer are as follows:

1. Open the Cloudera Manager Admin Console and go to the **Key-Value Store Indexer**.
2. Click the **Configuration** tab.
3. Select **Scope** > **All**.
4. Select **Category** > **All**.
5. In the Search field, type **TLS/SSL** to show the Solr TLS/SSL properties.
6. Edit the following TLS/SSL properties according to your cluster configuration.

> **Note:** These values must be the same for all hosts running the Key-Value Store Indexer role.

Table 17: Key-Value Store TLS/SSL Properties

Property	Description
HBase Indexer TLS/SSL Certificate Trust Store File	The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that HBase Indexer might connect to. This is used when HBase Indexer is the client in a TLS/SSL connection. This truststore must

Property	Description
	contain the certificate(s) used to sign the service(s) being connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
HBase Indexer TLS/SSL Certificate Trust Store Password (Optional)	The password for the HBase Indexer TLS/SSL Certificate Trust Store File. This password is not required to access the truststore: this field can be left blank. This password provides optional integrity checking of the file. The contents of truststores are certificates, and certificates are public information.

7. Restart the service.

Configuring TLS/SSL for the Key-Value Store Indexer Using the Command Line

For every host running Key-Value Store Indexer server, specify Solr Trust Store details using the `HBASE_INDEXER_OPTS` environmental variable using following Java system properties:

- `-Djavax.net.ssl.trustStore=<absolute_path_to_truststore_file>`
- `-Djavax.net.ssl.trustStorePassword=<truststore_password>` (Optional)

Restart the Key-Value Store Indexer servers to apply these changes.

Configuring TLS/SSL for Flume Using Cloudera Manager

The steps for configuring and enabling Hadoop TLS/SSL for Flume are as follows:

1. Open the Cloudera Manager Admin Console and go to **Flume**.
2. Click the **Configuration** tab.
3. Select **Scope** > **All**.
4. Select **Category** > **All**.
5. In the Search field, type **TLS/SSL** to show the properties.
6. Edit the following SSL properties according to your cluster configuration.

> **Note:** These values must be the same for all hosts running the Flume role.

Table 18: Key-Value Store SSL Properties

Property	Description
Flume TLS/SSL Certificate Trust Store File	The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that Flume might connect to. This is used when Flume is the client in a TLS/SSL connection. This truststore must contain the certificate(s) used to sign the service(s) being connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
Flume TLS/SSL Certificate Trust Store Password (Optional)	The password for the Flume TLS/SSL Certificate Trust Store File. This password is not required to access the truststore: this field can be left blank. This password provides optional integrity checking of the file. The contents of truststores are certificates, and certificates are public information.

7. Click **Save Changes** to commit the changes.
8. Restart the service.

Configuring TLS/SSL for Flume Using the Command Line

For every host running Flume agent, specify Solr Trust Store details using the `FLUME_AGENT_JAVA_OPTS` environmental variable using following Java system properties:

- `-Djavax.net.ssl.trustStore=<absolute_path_to_truststore_file>`
- `-Djavax.net.ssl.trustStorePassword=<truststore_password>` (Optional)

Restart the Flume agents to apply these changes.

Spark Encryption

Spark supports the following means of encrypting Spark data at rest, and data in transit.

Enabling Encrypted Shuffle for Spark on YARN

The following properties must be configured to enable encrypted shuffle for Spark on YARN. Spark does *not* support encryption for cached data or intermediate files that spill to the local disk.

To use Cloudera Manager to configure these properties, see Enabling Spark Encryption Using Cloudera Manager on page 218. To use the command line instead, add the properties listed here to `/etc/spark/conf/spark-defaults.conf` on the host that launches Spark jobs.

Property	Description
spark.shuffle.encryption.enabled	Enable encrypted communication when authentication is enabled. This option is currently only supported by the block transfer service.
spark.shuffle.encryption.keySizeBits	Shuffle file encryption key size in bits. The valid numbers include 128, 192, and 256.
spark.shuffle.encryption.keygen.algorithm	The algorithm to generate the key used by shuffle file encryption.
spark.shuffle.crypto.cipher.transformation	Cipher transformation for shuffle file encryption. Currently only AES/CTR/NoPadding is supported.
spark.shuffle.crypto.cipher.classes	Comma-separated list of crypto cipher classes that implement AES/CTR/NoPadding. A crypto cipher implementation encapsulates encryption and decryption details. The first available implementation in this list is used.
spark.shuffle.crypto.secure.random.classes	Comma-separated list of secure random classes that implement a secure random algorithm. Use this when generating the Initialization Vector for crypto input/output streams. The first available implementation in this list is used.

Enabling SASL Encryption for Spark RPCs

If you are using an external shuffle service, configure the following property in the shuffle service configuration to disable unencrypted connections. This setting will only work for connections from services that use SASL for authentication. Note that the external shuffle service is enabled by default in CDH 5.5 and higher.

Property	Default Value	Description
spark.network.sasl.serverAlwaysEncrypt	false	Disable unencrypted connections for the external shuffle service.

If you are using the block transfer service, configure the following property to enable SASL encryption for Spark RPCs. This setting is supported only when authentication using a secret key is already enabled.

Property	Default Value	Description
spark.authenticate.enableSaslEncryption	false	Enable encrypted communication for the block transfer service.

To use Cloudera Manager to configure these properties, see Enabling Spark Encryption Using Cloudera Manager on page 218. To use the command line instead, add the properties listed here to `/etc/spark/conf/spark-defaults.conf` on the host that launches Spark jobs.

Enabling Spark Encryption Using Cloudera Manager

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

1. Open the Cloudera Manager Admin Console and go to the **Spark** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Gateway**.
4. Select **Category** > **Advanced**.
5. Edit the **Spark Client Advanced Configuration Snippet (Safety Valve) for spark-conf/spark-defaults.conf** property and add configuration properties for the feature you want to enable.
6. Click **Save Changes** to commit the changes.
7. Restart the Spark service.

Configuring TLS/SSL for HttpFS

Important:

- This configuration process can be completed using either Cloudera Manager or the command-line instructions.
- This information applies specifically to CDH 5.13.x. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

Using Cloudera Manager

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

The steps for configuring and enabling TLS/SSL for HttpFS using Cloudera Manager are as follows:

1. Go to the **HDFS service**
2. Click the **Configuration** tab.
3. Select **Scope** > **HttpFS**.
4. **Select** > **Security**.
5. Edit the following TLS/SSL properties according to your cluster configuration:

Table 19: HttpFS TLS/SSL Properties

Property	Description
Use TLS/SSL	Use TLS/SSL for HttpFS.
HttpFS Keystore File	Location of the keystore file used by the HttpFS role for TLS/SSL. **Default:** `/var/run/hadoop-httpfs/.keystore`. Note that the default location for the keystore file is on non-persistent disk.
HttpFS Keystore Password	Password of the keystore used by the HttpFS role for TLS/SSL. If the keystore password has a percent sign, it must be escaped. For example, for a password that is `pass%word`, use `pass%%word`.
HttpFS TLS/SSL Certificate Trust Store File	The location on disk of the truststore, in `.jks` format, used to confirm the authenticity of TLS/SSL servers that HttpFS might connect to. This is used when HttpFS is the client in a TLS/SSL connection.

Property	Description
HttpFS TLS/SSL Certificate Trust Store Password	The password for the HttpFS TLS/SSL Certificate Trust Store File. This password is not required to access the truststore; this field can be left blank. If the truststore password has a percent sign, it must be escaped. For example, for a password that is `pass%word`, use `pass%%word`.

6. Click **Save Changes**.
7. Restart the HDFS service.

Connect to the HttpFS Web UI using TLS/SSL (HTTPS)

Use `https://<httpfs_server_hostname>:14000/webhdfs/v1/`, though most browsers should automatically redirect you if you use `http://<httpfs_server_hostname>:14000/webhdfs/v1/`

Using the Command Line

Configure the HttpFS Server to use TLS/SSL (HTTPS)

1. Stop HttpFS by running

```
sudo /sbin/service hadoop-httpfs stop
```

2. To enable TLS/SSL, change which configuration the HttpFS server should work with using the `alternatives` command.

> **Note:** The `alternatives` command is only available on RHEL systems. For SLES, Ubuntu and Debian systems, the command is `update-alternatives`.

For RHEL systems, to use TLS/SSL:

```
alternatives --set hadoop-httpfs-tomcat-conf /etc/hadoop-httpfs/tomcat-conf.https
```

> **Important:**
>
> The HTTPFS_TLS/SSL_KEYSTORE_PASS variable must be the same as the password used when creating the keystore file. If you used a password other than `password`, you'll have to change the value of the HTTPFS_TLS/SSL_KEYSTORE_PASS variable in `/etc/hadoop-httpfs/conf/httpfs-env.sh`.

3. Start HttpFS by running

```
sudo /sbin/service hadoop-httpfs start
```

Connect to the HttpFS Web UI using TLS/SSL (HTTPS)

Use `https://<httpfs_server_hostname>:14000/webhdfs/v1/`, though most browsers should automatically redirect you if you use `http://<httpfs_server_hostname>:14000/webhdfs/v1/`

> **Important:**
>
> If using a Self-Signed Certificate, your browser will warn you that it cannot verify the certificate or something similar. You will probably have to add your certificate as an exception.

Encrypted Shuffle and Encrypted Web UIs

> **Important:**
>
> - If you use Cloudera Manager, do not use these command-line instructions. For the Cloudera Manager instructions, see Configuring TLS/SSL for HDFS, YARN and MapReduce on page 198.
> - This information applies specifically to CDH 5.13.0. If you use a lower version of CDH, see the documentation for that version located at Cloudera Documentation.

CDH 5 supports encryption of the MapReduce shuffle phase for both MapReduce v1 (MRv1) and MapReduce v2 (MRv2), also known as YARN. CDH also supports enabling TLS/SSL for the MRv1 and YARN web UIs, with optional client authentication (also known as bi-directional HTTPS, or HTTPS with client certificates). The configuration properties required to enable these features have been combined. In most cases, these properties are common to both MRv1 and YARN. They include:

- `hadoop.ssl.enabled`:
  - Toggles the shuffle for MRv1 between HTTP and HTTPS.
  - Toggles the MRv1 and YARN web UIs between HTTP and HTTPS.
- `mapreduce.shuffle.ssl.enabled`: Toggles the shuffle for YARN between HTTP and HTTPS.

  By default, this property is not specified in `mapred-site.xml`, and YARN encrypted shuffle is controlled by the value of `hadoop.ssl.enabled`. If this property is set to `true`, encrypted shuffle is enabled for YARN. Note that you cannot successfully enable encrypted shuffle for YARN by only setting this property to `true`, if `hadoop.ssl.enabled` is still set to `false`.
- Configuration settings for specifying keystore and truststore properties that are used by the MapReduce shuffle service, the Reducer tasks that fetch shuffle data, and the web UIs.
- `ssl.server.truststore.reload.interval`: A configuration property to reload truststores across the cluster when a node is added or removed.

> **Important:**
>
> When the web UIs are served over HTTPS, you must specify `https://` as the protocol. There is no redirection from `http://`. If you attempt to access an HTTPS resource over HTTP, your browser will show an empty screen with no warning.

Configuring Encrypted Shuffle and Encrypted Web UIs

Configure encryption for the MapReduce shuffle, and the MRv1 and YARN web UIs, as follows:

Enable encrypted shuffle for MRv1, and encryption for the MRv1 and YARN web UIs (core-site.xml)

Set the following properties in the `core-site.xml` files of all nodes in the cluster.

`hadoop.ssl.enabled`

Default value: `false`

For MRv1, set this value to `true` to enable encryption for both the MapReduce shuffle and the web UI.

For YARN, this property enables encryption for the web UI only . Enable shuffle encryption with a property in the `mapred-site.xml` file as described here.

`hadoop.ssl.require.client.cert`

Default value: `false`

When this property is set to `true`, client certificates are required for all shuffle operations and all browsers used to access web UIs.

Cloudera recommends that this be set to `false`. This is because client certificates are easily susceptible to attacks from malicious clients or jobs. For more details, see Client Certificates on page 225.

`hadoop.ssl.hostname.verifier`

Default value: `DEFAULT`

The `SSLHostnameVerifier` interface present inside the hadoop-common security library checks if a hostname matches the name stored inside the server's X.509 certificate. The value assigned to this property determines how Hadoop verifies hostnames when it establishes new `HttpsURLConnection` instances. Valid values are:

- `DEFAULT`: The hostname must match either the first common name (CN) or any of the subjectAltNames (SAN). Wildcards can occur in either the CN or the SANs. For example, a hostname, such as `*.example.com`, will match all subdomains, including `test.cloudera.example.com`.
- `DEFAULT_AND_LOCALHOST`: This verifier mechanism works just like `DEFAULT`. However, it also allows all hostnames of the type: `localhost`, `localhost.example`, or `127.0.0.1`.
- `STRICT`: This verifier works just like `DEFAULT` with an additional restriction for hostnames with wildcards. For example, a hostname with a wildcard such as `*.example.com`, will only match subdomains at the same level. Hence, `cloudera.example.com` will match, but, unlike `DEFAULT`, `test.cloudera.example.com` will be rejected.
- `STRICT_IE6`: This verifier works just like `STRICT`, however, it will allow hostnames that match any of the common names (CN) within the server's X.509 certificate, not just the first one.
- `ALLOW_ALL`: Using this verifier will essentially turn off the hostname verifier mechanism.

`hadoop.ssl.keystores.factory.class`

Default value: `org.apache.hadoop.security.ssl.FileBasedKeyStoresFactory`

The `KeyStoresFactory` implementation to be used. Currently, `FileBasedKeyStoresFactory` is the only implementation of `KeyStoresFactory`.

`hadoop.ssl.server.conf`

Default value: `ssl-server.xml`

Resource file from which TLS/SSL server keystore information is extracted. Typically, it should be in the `/etc/hadoop/conf/` directory so that it can be looked up in the `CLASSPATH`.

`hadoop.ssl.client.conf`

Default value: `ssl-client.xml`

Resource file from which TLS/SSL client keystore information is extracted. Typically, it should be in the `/etc/hadoop/conf/` directory so that it can be looked up in the `CLASSPATH`.

Set the `<final>` field for all these properties to `true` as in the following sample configuration snippet:

```
...
    <property>
      <name>hadoop.ssl.require.client.cert</name>
      <value>false</value>
      <final>true</final>
    </property>

    <property>
      <name>hadoop.ssl.hostname.verifier</name>
      <value>DEFAULT</value>
      <final>true</final>
    </property>

    <property>
      <name>hadoop.ssl.keystores.factory.class</name>
      <value>org.apache.hadoop.security.ssl.FileBasedKeyStoresFactory</value>
      <final>true</final>
    </property>
```

```
    <property>
      <name>hadoop.ssl.server.conf</name>
      <value>ssl-server.xml</value>
      <final>true</final>
    </property>

    <property>
      <name>hadoop.ssl.client.conf</name>
      <value>ssl-client.xml</value>
      <final>true</final>
    </property>

    <property>
      <name>hadoop.ssl.enabled</name>
      <value>true</value>
    </property>
...
```

Enable encrypted shuffle for YARN (mapred-site.xml)

To enable encrypted shuffle for YARN, set the following property in the `mapred-site.xml` file on every node in the cluster:

`mapreduce.shuffle.ssl.enabled`

Default value: Not specified

By default, this property is not specified in `mapred-site.xml`, and YARN encrypted shuffle is controlled by the value of `hadoop.ssl.enabled`. If this property is set to `true`, encrypted shuffle is enabled for YARN. Note that you cannot successfully enable encrypted shuffle for YARN by only setting this property to `true`, if `hadoop.ssl.enabled` is still set to `false`.

Set the `<final>` field for this property to `true` as in the following configuration snippet:

```
...
    <property>
      <name>mapreduce.shuffle.ssl.enabled</name>
      <value>true</value>
      <final>true</final>
    </property>
...
```

Configure the keystore and truststore for the Shuffle server (ssl-server.xml)

Note: To run job tasks so they are prevented from reading the server keystore and gaining access to the shuffle server certificates:

- Configure the Linux Task Controller for MRv1
- Configure the Linux Container Executor for YARN

Currently, `FileBasedKeyStoresFactory` is the only implementation of `KeyStoresFactory`. It uses properties in the `ssl-server.xml` and `ssl-client.xml` files to configure the keystores and truststores.

The `ssl-server.xml` should be owned by the `hdfs` or `mapred` Hadoop system user, belong to the hadoop group, and it should have 440 permissions. Regular users should not belong to the hadoop group.

Use the following settings to configure the keystores and truststores in the `ssl-server.xml` file.

Property	Default Value	Description
`ssl.server.keystore.type`	`jks`	Keystore file type

Property	Default Value	Description
`ssl.server.keystore.location`	NONE	Keystore file location. The `mapred` user must own this file and have exclusive read access to it.
`ssl.server.keystore.password`	NONE	Keystore file password
`ssl.server.keystore.keypassword`	NONE	Key password
`ssl.server.truststore.type`	`jks`	Truststore file type
`ssl.server.truststore.location`	NONE	Truststore file location. The `mapred` user must own this file and have exclusive read access to it.
`ssl.server.truststore.password`	NONE	Truststore file password
`ssl.server.truststore.reload.interval`	10000	Truststore reload interval, in milliseconds

Sample `ssl-server.xml`

```
<configuration>
<!-- Server Certificate Store -->
<property>
  <name>ssl.server.keystore.type</name>
     <value>jks</value>
</property>
<property>
  <name>ssl.server.keystore.location</name>
  <value>${user.home}/keystores/server-keystore.jks</value>
</property>
<property>
  <name>ssl.server.keystore.password</name>
  <value>serverfoo</value>
</property>
<property>
  <name>ssl.server.keystore.keypassword</name>
  <value>serverfoo</value>
</property>

<!-- Server Truststore -->
<property>
  <name>ssl.server.truststore.type</name>
  <value>jks</value>
</property>
<property>
  <name>ssl.server.truststore.location</name>
  <value>${user.home}/keystores/truststore.jks</value>
</property>
<property>
  <name>ssl.server.truststore.password</name>
  <value>clientserverbar</value>
</property>
<property>
  <name>ssl.server.truststore.reload.interval</name>
  <value>10000</value>
</property>
</configuration>
```

Configure the keystore and truststore for the Reducer/Fetcher (ssl-client.xml)

Use the following settings to configure the keystore and truststore in the `ssl-client.xml` file. This file must be owned by the `mapred` user for MRv1 and by the `yarn` user for YARN. The file permissions should be 444 (read access for all users).

Property	Default Value	Description
ssl.client.keystore.type	jks	Keystore file type
ssl.client.keystore.location	NONE	Keystore file location. The mapred user must own this file and should have read access to it.
ssl.client.keystore.password	NONE	Keystore file password
ssl.client.keystore.keypassword	NONE	Key password
ssl.client.truststore.type	jks	Truststore file type
ssl.client.truststore.location	NONE	Truststore file location. The mapred user must own this file and should have read access to it.
ssl.client.truststore.password	NONE	Truststore file password
ssl.client.truststore.reload.interval	10000	Truststore reload interval, in milliseconds

Sample ssl-client.xml

```
<configuration>
  <!-- Client Certificate Store -->
  <property>
    <name>ssl.client.keystore.type</name>
    <value>jks</value>
  </property>
  <property>
    <name>ssl.client.keystore.location</name>
    <value>${user.home}/keystores/client-keystore.jks</value>
  </property>
  <property>
    <name>ssl.client.keystore.password</name>
    <value>clientfoo</value>
  </property>
  <property>
    <name>ssl.client.keystore.keypassword</name>
    <value>clientfoo</value>
  </property>

  <!-- Client Truststore -->
  <property>
    <name>ssl.client.truststore.type</name>
    <value>jks</value>
  </property>
  <property>
    <name>ssl.client.truststore.location</name>
    <value>${user.home}/keystores/truststore.jks</value>
  </property>
  <property>
    <name>ssl.client.truststore.password</name>
    <value>clientserverbar</value>
  </property>
  <property>
    <name>ssl.client.truststore.reload.interval</name>
    <value>10000</value>
  </property>
</configuration>
```

Activating Encrypted Shuffle

> **Important:**
>
> Encrypted shuffle has a significant performance impact. You should benchmark this before implementing it in production. In many cases, one or more additional cores are needed to maintain performance.

When you have made the configuration changes described in the previous section, activate Encrypted Shuffle by re-starting all TaskTrackers in MRv1 and all NodeManagers in YARN.

Client Certificates

Client Certificates are supported but they do not guarantee that the client is a reducer task for the job. The Client Certificate keystore file that contains the private key must be readable by all users who submit jobs to the cluster, which means that a rogue job could read those keystore files and use the client certificates in them to establish a secure connection with a Shuffle server. The JobToken mechanism that the Hadoop environment provides is a better protector of the data; each job uses its own JobToken to retrieve only the shuffle data that belongs to it. Unless the rogue job has a proper JobToken, it cannot retrieve Shuffle data from the Shuffle server.

However, if your cluster requires client certificates, ensure that browsers connecting to the web UIs are configured with appropriately signed certificates. If your certificates are signed by a certificate authority (CA), make sure you include the complete chain of CA certificates in the server's keystore.

Reloading Truststores

By default, each truststore reloads its configuration every 10 seconds. If you bring in a new truststore file to replace an old one, when the truststore is reloaded, the new certificates will be override the previous ones. If a client certificate is added to (or removed from) all the truststore files in the system, both YARN and MRv1 will pick up the new configuration without requiring that the TaskTracker or NodeManager daemons are restarted. This mechanism is useful for adding or removing nodes from the cluster, or for adding or removing trusted clients.

The reload interval is controlled by the `ssl.client.truststore.reload.interval` and `ssl.server.truststore.reload.interval` configuration properties in the `ssl-client.xml` and `ssl-server.xml` files described here.

> **Note:** The keystores are not automatically reloaded. To change a keystore for a TaskTracker in MRv1 or a NodeManager in YARN, you must restart the TaskTracker or NodeManager daemon.

Debugging

> **Important:** Enable debugging only for troubleshooting, and only for jobs running on small amounts of data. Debugging is very verbose and slows jobs down significantly. You may need to increase the value for the `mapred.task.timeout` property to prevent jobs from failing for taking too long.

To enable TLS/SSL debugging in the reducers, set the `mapred.reduce.child.java.opts` property as follows. You can do this on a per-job basis, or by means of a cluster-wide setting in `mapred-site.xml`.:

```
<configuration>
...
    <property>
        <name>mapred.reduce.child.java.opts</name>
        <value>-Xmx200m -Djavax.net.debug=all</value>
    </property>
...
</configuration>
```

To enable debugging for MRv1 TaskTrackers, edit `hadoop-env.sh` as follows:

```
HADOOP_TASKTRACKER_OPTS="-Djavax.net.debug=all $HADOOP_TASKTRACKER_OPTS"
```

To enable debugging for YARN NodeManagers for YARN, edit `yarn-env.sh` as follows:

```
YARN_OPTS="-Djavax.net.debug=all $YARN_OPTS"
```

Configuring TLS/SSL for Navigator Audit Server

Cloudera Navigator supports TLS/SSL encryption for network communications between the Navigator Audit Server and clients, such as the web browser used for Cloudera Navigator console. Typically, TLS/SSL is configured for the entire cluster, so it is possible that the server key and certificate already exist on the specific host running the Navigator Audit Server role because that role runs on the same host as Cloudera Management Server. See Configuring Cloudera Manager Clusters for TLS/SSL on page 183 for more information about configuring TLS/SSL for Cloudera Manager clusters.

1. Log in to the Cloudera Manager Admin Console.
2. Select **Clusters** > **Cloudera Management Service**.
3. Click the **Configuration** tab.
4. Select **Scope** > **Navigator Audit Server**.
5. Select **Category** > **Security**.
6. Edit the following properties according to your cluster configuration.

Property	Description
Enable TLS/SSL for Navigator Audit Server	Encrypt network communications between clients and Navigator Audit Server using TLS/SSL.
TLS/SSL Keystore File Location	The path to the keystore file containing the server private key and certificate. The keystore must be in JKS format.
TLS/SSL Keystore File Password	The password for the Navigator Audit Server JKS keystore file.
TLS/SSL Keystore Key Password	The password for the private key contained in the JKS keystore.

7. Click **Save Changes**.
8. Restart the Navigator Audit Server role.

Note: After TLS/SSL is enabled, Cloudera Manager links to the Cloudera Navigator console use HTTPS rather than unencrypted HTTP.

Configuring TLS/SSL for Navigator Metadata Server

Cloudera Navigator supports TLS/SSL encryption for network communications between the Navigator Metadata Server and clients, such as the web browser used for Cloudera Navigator console. Typically, TLS/SSL is configured for the entire cluster, so it is possible that the server key and certificate already exist on the specific host running the Navigator Metadata Server role. The assumption in the steps below is that the cluster is already configured for TLS/SSL and the security artifacts have already been obtained and deployed to the host running the Navigator Metadata Server role instance.

1. Log in to the Cloudera Manager Admin Console.
2. Select **Clusters** > **Cloudera Management Service**.
3. Click the **Configuration** tab.
4. Select **Scope** > **Navigator Metadata Server**.

5. Select **Category** > **Security**.
6. Edit the following properties according to your cluster configuration.

Property	Description
Enable TLS/SSL for Navigator Metadata Server	Encrypt network communications between clients and Navigator Metadata Server using TLS/SSL.
TLS/SSL Keystore File Location	The path to the keystore file containing the server private key and certificate. The keystore must be in JKS format.
TLS/SSL Keystore File Password	The password for the Navigator Metadata Server JKS keystore file.
TLS/SSL Keystore Key Password	The password for the private key contained in the JKS keystore.

7. Click **Save Changes**.
8. Restart the Navigator Metadata Server role.

> **Note:** After TLS/SSL is enabled, Cloudera Manager links to the Cloudera Navigator console use HTTPS rather than unencrypted HTTP.

Configuring TLS/SSL for Kafka (Navigator Event Broker)

To enable TLS/SSL encryption between Navigator Audit Server and Kafka for publishing audit events to Kafka:

1. Log in to the Cloudera Manager Admin Console.
2. Select **Clusters** > **Kafka**.
3. Click the **Configuration** tab.
4. Select **Kafka Broker** for the **Scope** filter.
5. Select **Security** for the **Category** filter.
6. Enter the following properties according to your cluster configuration.

Property	Description
Enable TLS/SSL for Kafka Broker	Select the checkbox to enable TLS/SSL for encrypted communication between clients and the Kafka Broker service.
Kafka Broker TLS/SSL Certificate Trust Store File	Enter the path (location on disk) to the JKS truststore. Leave this field empty to have the list of well-known CAs checked to provide a chain of proof for the Navigator Audit Server.
Kafka Broker TLS/SSL Certificate Trust Store Password	The truststore does not need password protection. Its contents are public certificates already included in the default Java truststore.

7. Click **Save Changes**.
8. Restart the Kafka service.

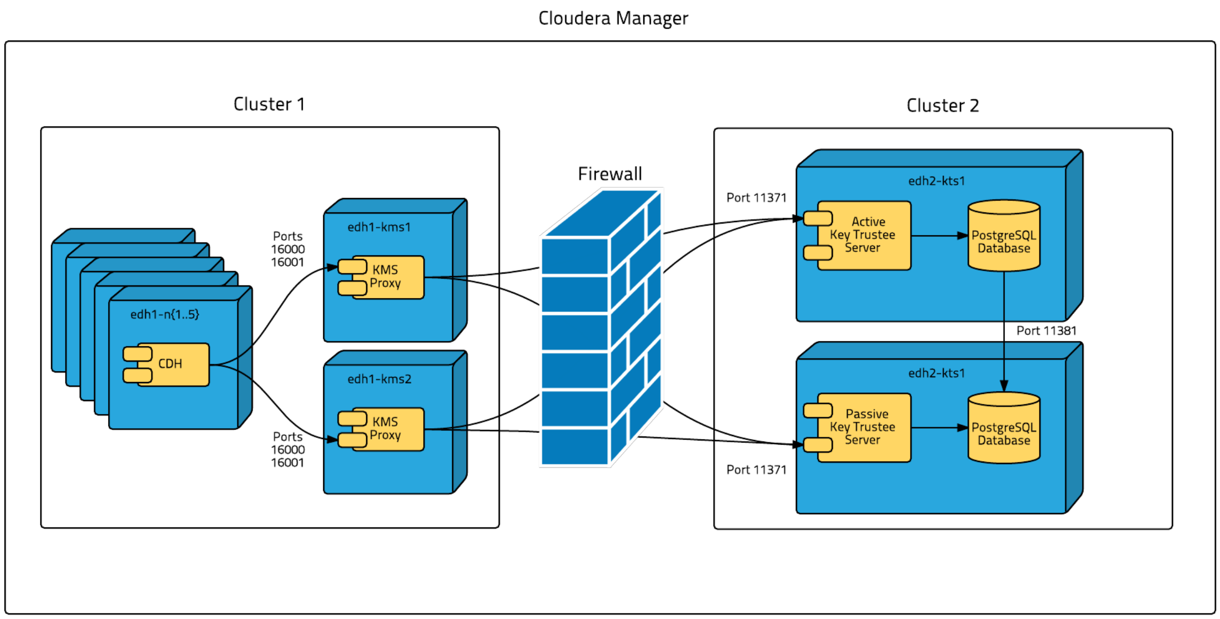
Data at Rest Encryption

Cloudera clusters can use a combination of data at rest encryption mechanisms, including HDFS transparent encryption and Cloudera Navigator Encrypt. Both of these data at rest encryption mechanisms can be augmented with key management using Cloudera Navigator Key Trustee Server on page 283 and Cloudera Navigator Key HSM on page 301.

Before configuring encryption for data at rest, see Cloudera Navigator Data Encryption Overview for an overview of Cloudera Navigator concepts, architecture, and components.

Data at Rest Encryption Reference Architecture

The following diagram illustrates the supported architecture for deploying Cloudera Navigator encryption for data at rest:



To isolate Key Trustee Server from other enterprise data hub (EDH) services, you must deploy Key Trustee Server on dedicated hosts in a separate cluster in Cloudera Manager. Deploy Key Trustee KMS on dedicated hosts in the same cluster as the EDH services that require access to Key Trustee Server. This provides the following benefits:

- You can restart your EDH cluster without restarting Key Trustee Server, avoiding interruption to other clusters or clients that use the same Key Trustee Server instance.
- You can manage the Key Trustee Server upgrade cycle independently of other cluster components.
- You can limit access to the Key Trustee Server hosts to authorized key administrators only, reducing the attack surface of the system.
- Resource contention is reduced. Running Key Trustee Server and Key Trustee KMS services on dedicated hosts prevents other cluster services from reducing available resources (such as CPU and memory) and creating bottlenecks.

If you are using virtual machines for the Key Trustee Server or Key Trustee KMS hosts, see Navigator HSM KMS HA Planning on page 234.

Data at Rest Encryption Requirements

Encryption comprises several components, each with its own requirements.

Data at rest encryption protection can be applied at a number of levels within Hadoop:

- OS filesystem-level
- Network-level
- HDFS-level (protects both data at rest and in transit)

For more information on the components, concepts, and architecture for encrypting data at rest, see Data at Rest Encryption on page 228.

Product Compatibility Matrix

See Product Compatibility Matrix for Cloudera Navigator Encryption for the individual compatibility matrices for each Cloudera Navigator encryption component.

Entropy Requirements

Cryptographic operations require entropy to ensure randomness.

You can check the available entropy on a Linux system by running the following command:

```
$ cat /proc/sys/kernel/random/entropy_avail
```

The output displays the entropy currently available. Check the entropy several times to determine the state of the entropy pool on the system. If the entropy is consistently low (500 or less), you must increase it by installing `rng-tools` and starting the `rngd` service. Run the following commands on RHEL 6-compatible systems:

```
$ sudo yum install rng-tools
$ sudo echo 'EXTRAOPTIONS="-r /dev/urandom"' >> /etc/sysconfig/rngd
$ sudo service rngd start
$ sudo chkconfig rngd on
```

For RHEL 7, run the following commands:

```
$ sudo yum install rng-tools
$ cp /usr/lib/systemd/system/rngd.service /etc/systemd/system/
$ sed -i -e 's/ExecStart=\/sbin\/rngd -f/ExecStart=\/sbin\/rngd -f -r \/dev\/urandom/'
 /etc/systemd/system/rngd.service
$ systemctl daemon-reload
$ systemctl start rngd
$ systemctl enable rngd
```

Make sure that the hosts running Key Trustee Server, Key Trustee KMS, and Navigator Encrypt have sufficient entropy to perform cryptographic operations.

Key Trustee Server Requirements

Recommended Hardware and Supported Distributions

Key Trustee Server must be installed on a dedicated server or virtual machine (VM) that is not used for any other purpose. The backing PostgreSQL database must be installed on the same host as the Key Trustee Server, and must not be shared with any other services. For high availability, the active and passive Key Trustee Servers must not share physical resources. See Resource Planning for Data at Rest Encryption on page 233 for more information.

The recommended minimum hardware specifications are as follows:

- Processor: 1 GHz 64-bit quad core
- Memory: 8 GB RAM
- Storage: 20 GB on moderate- to high-performance disk drives

For information on the supported Linux distributions, see Table 11.

Cloudera Manager Requirements

Installing and managing Key Trustee Server using Cloudera Manager requires Cloudera Manager 5.4.0 and higher. Key Trustee Server does not require Cloudera Navigator Audit Server or Metadata Server.

SELinux Requirements

SELinux must be disabled for Key Trustee installation and operation. Modify `/etc/selinux/config` to set `SELINUX=disabled` and reboot the system for the change to take effect.

umask Requirements

Key Trustee Server installation requires the default `umask` of `0022`.

Network Requirements

For new Key Trustee Server installations (5.4.0 and higher) and migrated upgrades (see Migrate Apache Web Server to CherryPy for more information), Key Trustee Server requires the following TCP ports to be opened for inbound traffic:

- 11371

  Clients connect to this port over HTTPS.

- 11381 (PostgreSQL)

  The passive Key Trustee Server connects to this port for database replication.

For upgrades that are not migrated to the CherryPy web server, the pre-upgrade port settings are preserved:

- 80

  Clients connect to this port over HTTP to obtain the Key Trustee Server public key.

- 443 (HTTPS)

  Clients connect to this port over HTTPS.

- 5432 (PostgreSQL)

  The passive Key Trustee Server connects to this port for database replication.

TLS Certificate Requirements

To ensure secure network traffic, Cloudera recommends obtaining Transport Layer Security (TLS) certificates specific to the hostname of your Key Trustee Server. To obtain the certificate, generate a Certificate Signing Request (CSR) for the fully qualified domain name (FQDN) of the Key Trustee Server host. The CSR must be signed by a trusted Certificate Authority (CA). After the certificate has been verified and signed by the CA, the Key Trustee Server TLS configuration requires:

- The CA-signed certificate
- The private key used to generate the original CSR
- The intermediate certificate/chain file (provided by the CA)

Cloudera recommends not using self-signed certificates. If you use self-signed certificates, you must use the `--skip-ssl-check` parameter when registering Navigator Encrypt with the Key Trustee Server. This skips TLS hostname validation, which safeguards against certain network-level attacks. For more information regarding insecure mode, see Table 26: Registration Options on page 310.

Key Trustee KMS Requirements

Recommended Hardware and Supported Distributions

The recommended minimum hardware specifications are as follows:

- Processor: 2 GHz 64-bit quad core
- Memory: 16 GB RAM
- Storage: 40 GB on moderate- to high-performance disk drives

For information on the supported Linux distributions, see Table 12.

The Key Trustee KMS workload is CPU-intensive. Cloudera recommends using machines with capabilities equivalent to your NameNode hosts, with Intel CPUs that support AES-NI for optimum performance. Also, Cloudera strongly recommends that you enable TLS for both the HDFS and the Key Trustee KMS services to prevent the passage of plain text key material between the KMS and HDFS data nodes.

Key HSM Requirements

The following are prerequisites for installing Navigator Key HSM:

- Oracle Java Runtime Environment (JRE) 7 or higher with Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files:
  - JCE for Java SE 7
  - JCE for Java SE 8
- A supported Linux distribution. See Table 14.
- A supported HSM device:
  - SafeNet Luna
    - HSM firmware version: 6.2.1
    - HSM software version: 5.2.3-1
  - SafeNet KeySecure
    - HSM firmware version: 6.2.1
    - HSM software version: 8.0.1
  - Thales nSolo, nConnect
    - HSM firmware version: 11.4.0
    - Client software version: 2.28.9cam136
- Key Trustee Server 3.8 or higher

> **Important:** You must install Key HSM on the same host as Key Trustee Server.

Root access is required to install Navigator Key HSM.

Navigator HSM KMS Requirements

Recommended Hardware and Supported Distributions

The recommended minimum hardware specifications are as follows:

- Processor: 2 GHz 64-bit quad core
- Memory: 16 GB RAM
- Storage: 40 GB on moderate- to high-performance disk drives

Navigator HSM KMS supports the following Linux distributions:

- RHEL and CentOS: 6.8, 7.1, 7.2, 7.3

Supported HSM devices:

- SafeNet Luna
  - HSM software version: 6.2.2-5
  - HSM firmware version: 6.10.9
  - Client: 6.2.2
- Thales nSolo, nConnect
  - Server version: 3.67.11cam4
  - Firmware: 2.65.2
  - Security World Version: 12.30

Navigator Encrypt Requirements

Operating System Requirements

- Linux kernel 2.6.19 or higher (RHEL and CentOS can use 2.6.18-92 or higher)
- For supported Linux distributions, see Table 15.

> **Important:** With the exception of Cloudera Navigator Encrypt, Cloudera Enterprise is supported on platforms with Security-Enhanced Linux (SELinux) enabled. However, policies need to be provided by other parties or created by the administrator of the cluster deployment. Cloudera is not responsible for policy support nor policy enforcement, nor for any issues with them. If you experience issues with SELinux, contact your OS support provider.

Supported command-line interpreters:

- `sh` (Bourne)
- `bash` (Bash)
- `dash` (Debian)

> **Note:** Navigator Encrypt does not support installation or use in `chroot` environments.

SELinux Requirements

SELinux must be disabled for Key Trustee installation and operation. Modify `/etc/selinux/config` to set `SELINUX=disabled` and reboot the system for the change to take effect.

Network Requirements

For new Navigator Key Trustee Server (5.4.0 and higher) installations, Navigator Encrypt initiates TCP traffic over port 11371 (HTTPS) to the Key Trustee Server.

For upgrades and Key Trustee Server versions lower than 5.4.0, Navigator Encrypt initiates TCP traffic over ports 80 (HTTP) and 443 (HTTPS) to the Navigator Key Trustee Server.

Internet Access

You must have an active connection to the Internet to download many package dependencies, unless you have internal repositories or mirrors containing the dependent packages.

Maintenance Window

Data is not accessible during the encryption process. Plan for system downtime during installation and configuration.

Administrative Access

To enforce a high level of security, all Navigator Encrypt commands require administrative (root) access (including installation and configuration). If you do not have administrative privileges on your server, contact your system administrator before proceeding.

Package Dependencies

Navigator Encrypt requires these packages, which are resolved by your distribution package manager during installation:

- `dkms`
- `keyutils`
- `ecryptfs-utils`
- `libkeytrustee`
- `navencrypt-kernel-module`
- `openssl`
- `lsof`
- `gcc`
- `cryptsetup`

These packages may have other dependencies that are also resolved by your package manager. Installation works with `gcc`, `gcc3`, and `gcc4`.

Resource Planning for Data at Rest Encryption

For production environments, you must configure high availability for:

- Key Trustee Server
- Key Trustee KMS
- Navigator HSM KMS

Key Trustee Server and Key Trustee KMS HA Planning

For high availability, you must provision two dedicated Key Trustee Server hosts and at least two dedicated Key Trustee KMS hosts, for a minimum of four separate hosts. Do not run multiple Key Trustee Server or Key Trustee KMS services on the same physical host, and do not run these services on hosts with other cluster services. Doing so causes resource contention with other important cluster services and defeats the purpose of high availability. See Data at Rest Encryption Reference Architecture on page 228 for more information.

The Key Trustee KMS workload is CPU intensive. Cloudera recommends using machines with capabilities equivalent to your NameNode hosts, with Intel CPUs that support AES-NI for optimum performance.

Make sure that each host is secured and audited. Only authorized key administrators should have access to them. Red Hat provides security guides for RHEL:

- RHEL 6 Security Guide
- RHEL 7 Security Guide

For hardware sizing information, see Data at Rest Encryption Requirements on page 229 for recommendations for each Cloudera Navigator encryption component.

For Cloudera Manager deployments, deploy Key Trustee Server in its own dedicated cluster. Deploy Key Trustee KMS in each cluster that uses Key Trustee Server. See Data at Rest Encryption Reference Architecture on page 228 for more information.

For information about enabling Key Trustee Server high availability, refer to Configuring Key Trustee Server High Availability Using Cloudera Manager or Configuring Key Trustee Server High Availability Using the Command Line.

For information about enabling Key Trustee KMS high availability, refer to Enabling Key Trustee KMS High Availability.

Navigator HSM KMS HA Planning

For Navigator HSM KMS high availability, you need to provision two dedicated HSM KMS hosts only.

Make sure that each host is secured and audited. Only authorized key administrators should have access to them. Red Hat provides security guides for RHEL:

- RHEL 6 Security Guide
- RHEL 7 Security Guide

For hardware sizing information, see Data at Rest Encryption Requirements on page 229 for recommendations for each Cloudera Navigator encryption component.

For information about enabling HSM KMS high availability, refer to Enabling Navigator HSM KMS High Availability.

Virtual Machine Considerations

If you are using virtual machines, make sure that the resources (such as virtual disks, CPU, and memory) for each Key Trustee Server and Key Trustee KMS host are allocated to separate physical hosts. Hosting multiple services on the same physical host defeats the purpose of high availability, because a single machine failure can take down multiple services.

To maintain the security of the cryptographic keys, make sure that all copies of the virtual disk (including any back-end storage arrays, backups, snapshots, and so on) are secured and audited with the same standards you apply to the live data.

HDFS Transparent Encryption

Data encryption is mandatory for many government, financial, and regulatory entities, worldwide, to meet privacy and other security requirements. For example, the card payment industry has adopted the Payment Card Industry Data Security Standard (PCI DSS) for information security. Other examples include requirements imposed by United States government's Federal Information Security Management Act (FISMA) and Health Insurance Portability and Accountability Act (HIPAA). Encrypting data stored in HDFS can help your organization comply with such regulations.

Introduced in CDH 5.3, transparent encryption for HDFS implements transparent, end-to-end encryption of data read from and written to HDFS blocks across your cluster. *Transparent* means that end-users are unaware of the encryption/decryption processes, and *end-to-end* means that data is encrypted at-rest and in-transit (see the Cloudera Engineering Blog post for complete details).

> **Note:** HDFS Transparent Encryption is not the same as TLS encryption. Clusters configured using Level 3 TLS/SSL encrypt network communications throughout the cluster. Depending on the type of services your cluster supports, you may want to configure both HDFS Transparent Encryption and TLS/SSL for the cluster. See Configuring Encryption for more information.

HDFS encryption has these capabilities:

- Only HDFS clients can encrypt or decrypt data.
- Key management is external to HDFS. HDFS cannot access unencrypted data or encryption keys. Administration of HDFS and administration of keys are separate duties encompassed by distinct user roles (HDFS administrator, Key Administrator), thus ensuring that no single user has unrestricted access to both data and keys.
- The operating system and HDFS interact using encrypted HDFS data only, mitigating threats at the OS- and file-system-level.
- HDFS uses the Advanced Encryption Standard-Counter mode (AES-CTR) encryption algorithm. AES-CTR supports a 128-bit encryption key (default), or can support a 256-bit encryption key when Java Cryptography Extension (JCE) unlimited strength JCE is installed.
- HDFS encryption has been designed to take advantage of the AES-NI instruction set, a hardware-based encryption acceleration technique, so your cluster performance should not adversely affected by configuring encryption. (The AES-NI instruction set can be an order of magnitude faster than software implementations of AES.) However, you may need to update cryptography libraries on your HDFS and MapReduce client hosts to use the acceleration mechanism. See Optimizing Performance for HDFS Transparent Encryption on page 239 for details.

Key Concepts and Architecture

Keystores and the Hadoop Key Management Server

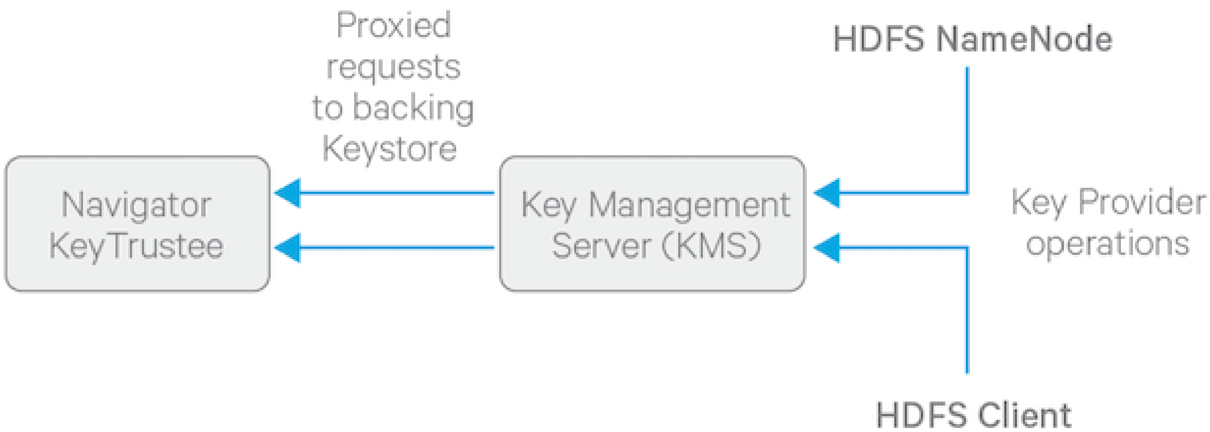
Integrating HDFS with an *external*, enterprise-level keystore is the first step to deploying transparent encryption. This is because separation of duties between a key administrator and an HDFS administrator is a very important aspect of this feature. However, most keystores are not designed for the encrypt/decrypt request rates seen by Hadoop workloads. This led to the development of a new service, called the **Hadoop Key Management Server (KMS)**, which serves as a proxy between HDFS clients and the backing keystore. Both the keystore and Hadoop KMS must use Hadoop's KeyProvider API to interact with each other and with HDFS clients.

While HDFS encryption can be used with a local Java KeyStore for key management, Cloudera does not recommend this for production environments where a more robust and secure key management solution should be used. Cloudera offers the following two options for enterprise-grade key management:

- Cloudera Navigator Key Trustee Server is a key store for managing encryption keys. To integrate with the Navigator Key Trustee Server, Cloudera provides a custom KMS service, Key Trustee KMS.

- Hardware security modules (HSM) are third-party appliances that provide the highest level of security for keys. To integrate with a list of supported HSMs, Cloudera provides a custom KMS service, Navigator HSM KMS (see Installing Navigator HSM KMS Backed by Thales HSM and Installing Navigator HSM KMS Backed by Luna HSM).

The diagram below illustrates how HDFS clients and the NameNode interact with an enterprise keystore through the Hadoop Key Management Server. The keystore can be either the Cloudera Navigator Key Trustee Server or a support HSM.



To get started with deploying the KMS and a keystore, see Enabling HDFS Encryption Using the Wizard on page 240.

For information on configuring and securing the KMS, see Configuring the Key Management Server (KMS) on page 259 and Securing the Key Management Server (KMS) on page 263.

Encryption Zones and Keys

HDFS transparent encryption introduces the concept of an ***encryption zone*** (EZ), which is a directory in HDFS whose contents will be automatically encrypted on write and decrypted on read. Encryption zones always start off as empty directories, and tools such as `distcp` can be used to add data to a zone. Every file and subdirectory copied to an encryption zone will be encrypted. You can rename directories and files; however, you *cannot* rename the encryption zone.

> **Note:** An encryption zone cannot be created on top of an existing directory.

Each encryption zone is associated with a key (EZ Key) specified by the key administrator when the zone is created. EZ keys are stored on a backing keystore external to HDFS. Each file within an encryption zone has its own encryption key, called the Data Encryption Key (DEK). These DEKs are encrypted with their respective encryption zone's EZ key, to form an Encrypted Data Encryption Key (EDEK).

The following diagram illustrates how encryption zone keys (EZ keys), data encryption keys (DEKs), and encrypted data encryption keys (EDEKs) are used to encrypt and decrypt files.

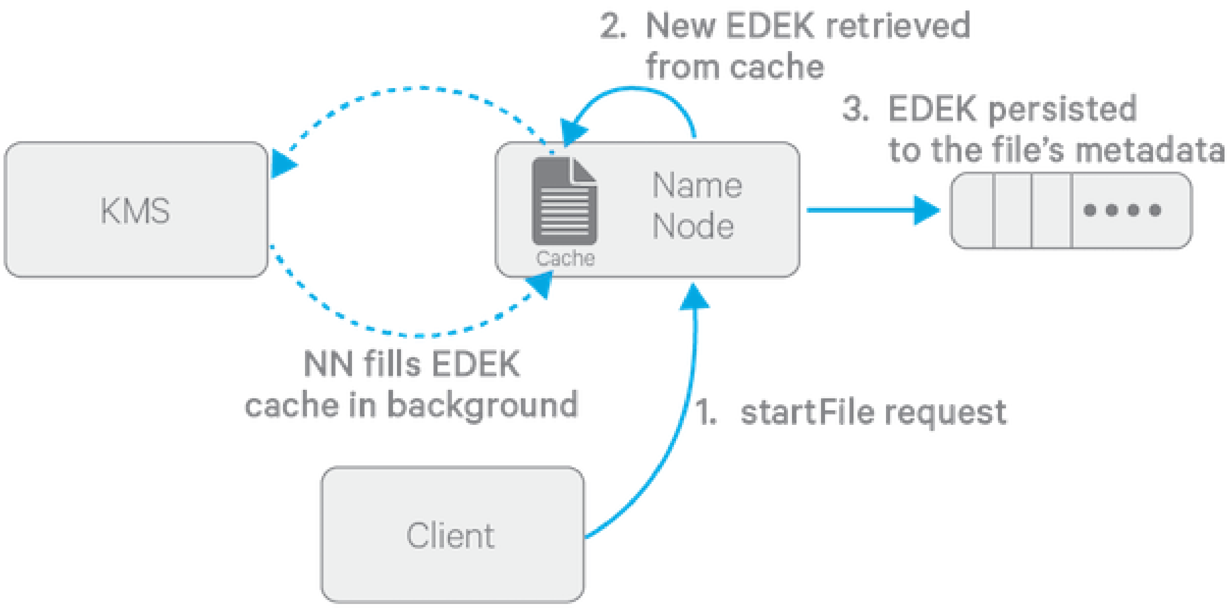
EDEKs are stored persistently on the NameNode as part of each file's metadata, using HDFS extended attributes (implemented as of Cloudera CDH 5.2). EDEKs can be safely stored and handled by the NameNode because the `hdfs` user does not have access to the EDEK's encryption keys (EZ keys). Even if HDFS is compromised (for example, by gaining unauthorized access to a superuser account), a malicious user only gains access to the encrypted text and EDEKs. EZ keys are controlled by a separate set of permissions on the KMS and the keystore.

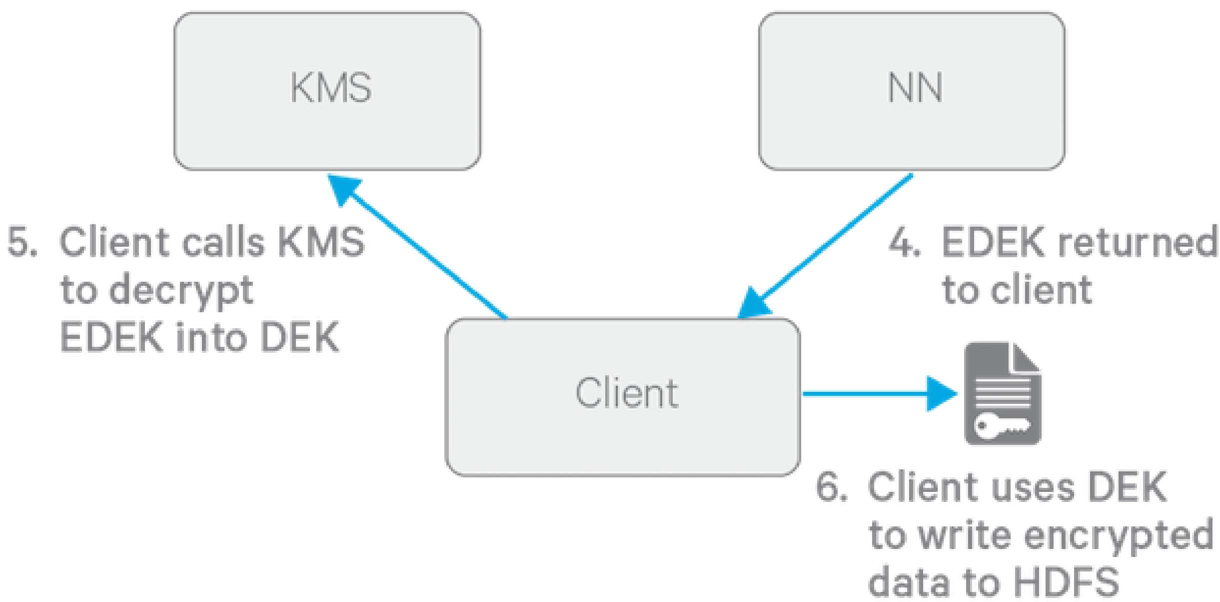
An EZ key can have multiple key versions, where each key version has its own distinct key material (that is, the portion of the key used during encryption and decryption). Key rotation is achieved by bumping up the version for an EZ key. Per-file key rotation is then achieved by re-encrypting the file's DEK with the new version of the EZ key to create new EDEKs. HDFS clients can identify an encryption key either by its key name, which returns the latest version of the key, or by a specific key version.

For more information on creating and managing encryption zones, see Managing Encryption Keys and Zones on page 250.

Accessing Files Within an Encryption Zone

To encrypt a new file, the HDFS client requests a new EDEK from the NameNode. The NameNode then asks the KMS to decrypt it with the encryption zone's EZ key. This decryption results in a DEK, which is used to encrypt the file.





The diagram above depicts the process of writing a new encrypted file. Note that the EDEK cache on the NameNode is populated in the background. Since it is the responsibility of KMS to create EDEKs, using a cache avoids having to call the KMS for each create request. The client can request new EDEKs directly from the NameNode.

To decrypt a file, the HDFS client provides the NameNode with the file's EDEK and the version number of the EZ key that was used to generate the EDEK. The NameNode requests the KMS to decrypt the file's EDEK with the encryption zone's EZ key, which involves checking that the requesting client has permission to access that particular version of the EZ key. Assuming decryption of the EDEK is successful, the client then uses this DEK to decrypt the file.

Encryption and decryption of EDEKs takes place entirely on the KMS. More importantly, the client requesting creation or decryption of an EDEK never handles the EZ key. Only the KMS can use EZ keys to create and decrypt EDEKs as requested. It is important to note that the KMS does not store any keys, other than temporarily in its cache. It is up to the enterprise keystore to be the authoritative storage for keys, and to ensure that keys are never lost, as a lost key is equivalent to introducing a security hole. For production use, Cloudera recommends you deploy two or more redundant enterprise key stores.

Optimizing Performance for HDFS Transparent Encryption

> **Warning:** To ensure that HDFS encryption functions as expected, the steps described in this section are *mandatory for production use*.

CDH implements the ***Advanced Encryption Standard New Instructions*** (AES-NI), which provide substantial performance improvements. To get these improvements, you need a recent version of `libcrypto.so` on HDFS and MapReduce client hosts -- that is, any host from which you originate HDFS or MapReduce requests. Many OS versions have an older version of the library that does not support AES-NI. The instructions that follow tell you what you need to do for each OS version that CDH supports.

RHEL/CentOS 6.5 or later

The installed version of `libcrypto.so` supports AES-NI, but you need to install the `openssl-devel` package on all clients:

```
$ sudo yum install openssl-devel
```

RHEL/CentOS 6.4 or earlier 6.x versions, or SLES 11

Download and extract a newer version of `libcrypto.so` from a CentOS 6.5 repository and install it on all clients in `/var/lib/hadoop/extra/native/`:

1. Download the latest version of the `openssl` package. For example:

```
$ wget
http://mirror.centos.org/centos/6/os/x86_64/Packages/openssl-1.0.1e-30.el6.x86_64.rpm
```

   The `libcrypto.so` file in this package can be used on SLES 11 as well as RHEL/CentOS.
2. Decompress the files in the package, but **do not** install it:

```
$ rpm2cpio openssl-1.0.1e-30.el6.x86_64.rpm | cpio -idmv
```

3. If you are using parcels, create the `/var/lib/hadoop/extra/native/` directory:

```
$ sudo mkdir -p /var/lib/hadoop/extra/native
```

4. Copy the shared library into `/var/lib/hadoop/extra/native/`. Name the target file `libcrypto.so`, with no suffix at the end, exactly as in the command that follows.

```
$ sudo cp ./usr/lib64/libcrypto.so.1.0.1e /var/lib/hadoop/extra/native/libcrypto.so
```

RHEL/CentOS 5

In this case, you need to build `libcrypto.so` and copy it to all clients:

1. On one client, compile and install `openssl` from source:

```
$ wget http://www.openssl.org/source/openssl-1.0.1j.tar.gz
$ cd openssl-1.0.1j
$ ./config --shared --prefix=/opt/openssl-1.0.1j
$ sudo make install
```

2. If you are using parcels, create the `/var/lib/hadoop/extra/native/` directory:

```
$ sudo mkdir -p /var/lib/hadoop/extra/native
```

3. Copy the files into /var/lib/hadoop/extra/native/:

```
$ sudo cp /opt/openssl-1.0.1j/lib/libcrypto.so /var/lib/hadoop/extra/native
```

4. Copy the files to the remaining clients using a utility such as rsync

Debian Wheezy

The installed version of libcrypto.so supports AES-NI, but you need to install the libssl-devel package on all clients:

```
$ sudo apt-get install libssl-dev
```

Ubuntu Precise and Ubuntu Trusty

Install the libssl-devel package on all clients:

```
$ sudo apt-get install libssl-dev
```

Testing if encryption optimization works

To verify that a client host is ready to use the AES-NI instruction set optimization for HDFS encryption at rest, use the following command:

```
hadoop checknative
```

You should see a response such as the following:

```
14/12/12 13:48:39 INFO bzip2.Bzip2Factory: Successfully loaded & initialized native-bzip2
library system-native14/12/12 13:48:39 INFO zlib.ZlibFactory: Successfully loaded &
initialized native-zlib library
Native library checking:
hadoop:  true /usr/lib/hadoop/lib/native/libhadoop.so.1.0.0
zlib:    true /lib64/libz.so.1
snappy:  true /usr/lib64/libsnappy.so.1
lz4:     true revision:99
bzip2:   true /lib64/libbz2.so.1
openssl: true /usr/lib64/libcrypto.so
```

If you see true in the openssl row, Hadoop has detected the right version of libcrypto.so and optimization will work. If you see false in this row, you do not have the right version.

Enabling HDFS Encryption Using the Wizard

To accommodate the security best practice of separation of duties, enabling HDFS encryption using the wizard requires different Cloudera Manager user roles for different steps.

Launch the **Set up HDFS Data At Rest Encryption** wizard in one of the following ways:

- **Cluster** > **Set up HDFS Data At Rest Encryption**

  Minimum Required Role: **Key Administrator** *or* **Cluster Administrator** (also provided by Full Administrator)

- **Administration** > **Security** > **Set up HDFS Data At Rest Encryption**

  Minimum Required Role: **Key Administrator** *or* **Cluster Administrator** (also provided by Full Administrator)

- **HDFS service** > **Actions** > **Set up HDFS Data At Rest Encryption**

  Minimum Required Role: **Cluster Administrator** (also provided by **Full Administrator**)

On the first page of the wizard, select the root of trust for encryption keys:

- **Cloudera Navigator Key Trustee Server**

- **Navigator HSM KMS backed by Thales HSM**
- **Navigator HSM KMS backed by Luna HSM**
- **A file-based password-protected Java KeyStore**

Cloudera strongly recommends using Cloudera Navigator Key Trustee Server as the root of trust for production environments. The file-based Java KeyStore root of trust is insufficient to provide the security, scalability, and manageability required by most production systems.

Choosing a root of trust displays a list of steps required to enable HDFS encryption using that root of trust. Each step can be completed independently. The **Status** column indicates whether the step has been completed, and the **Notes** column provides additional context for the step. If your Cloudera Manager user account does not have sufficient privileges to complete a step, the **Notes** column indicates the required privileges.

Available steps contain links to wizards or documentation required to complete the step. If a step is unavailable due to insufficient privileges or a prerequisite step being incomplete, no links are present and the **Notes** column indicates the reason the step is unavailable.

Continue to the section for your selected root of trust for further instructions:

Enabling HDFS Encryption Using Navigator Key Trustee Server

Enabling HDFS encryption using Key Trustee Server as the key store involves multiple components. For an overview of the components involved in encrypting data at rest, see Cloudera Navigator Data Encryption Overview. For guidelines on deploying the Navigator Key Trustee Server in production environments, Resource Planning for Data at Rest Encryption on page 233.

Before continuing, make sure the Cloudera Manager server host has access to the internal repository hosting the Key Trustee Server software. See Setting Up an Internal Repository for more information.

After selecting **Cloudera Navigator Key Trustee Server** as the root of trust, the following steps are displayed:

1. Enable Kerberos

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

For more information about enabling Kerberos, see Enabling Kerberos Authentication Using the Wizard on page 50.

2. Enable TLS/SSL

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

For more information about enabling TLS, see Configuring Cloudera Manager Clusters for TLS/SSL on page 183.

3. Add a dedicated cluster for the Key Trustee Server

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

This step creates a new cluster in Cloudera Manager for the Key Trustee Server hosts to isolate them from other enterprise data hub (EDH) services for increased security and durability. For more information, see Data at Rest Encryption Reference Architecture on page 228.

To complete this step:

1. Click **Add a dedicated cluster for the Key Trustee Server**.
2. Leave **Enable High Availability** checked to add two hosts to the cluster. For production environments, you must enable high availability for Key Trustee Server. Failure to enable high availability can result in complete data loss in the case of catastrophic failure of a standalone Key Trustee Server. Click **Continue**.
3. Search for new hosts to add to the cluster, or select the **Currently Managed Hosts** tab to add existing hosts to the cluster. After selecting the hosts, click **Continue**.
4. Select the KEYTRUSTEE_SERVER parcel to install Key Trustee Server using parcels, or select **None** if you want to use packages. If you do not see a parcel available, click **More Options** and add the repository URL to the **Remote Parcel Repository URLs** list. After selecting a parcel or **None**, click **Continue**.

If you selected **None**, click **Continue** again, and skip to 4. Install Key Trustee Server binary using packages or parcels on page 242.

5. After the KEYTRUSTEE_SERVER parcel is successfully downloaded, distributed, unpacked, and activated, click **Continue**.
6. Click **Continue** to complete this step and return to the main page of the wizard.

4. Install Key Trustee Server binary using packages or parcels

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

> **Note:** If you selected **None** on the parcel selection page in step 3. Add a dedicated cluster for the Key Trustee Server on page 241, the step title is changed to **Install Parcel for Key Trustee Server**. If you are using packages, skip this step and see Installing Key Trustee Server Using the Command Line for package-based installation instructions. After installing Key Trustee Server using packages, continue to 5. Install Parcel for Key Trustee KMS on page 242.

This step is completed automatically during 3. Add a dedicated cluster for the Key Trustee Server on page 241 if you are using parcels. If the step is incomplete for any reason (such as the wizard being interrupted or a failure installing the parcel), complete it manually:

1. Click **Install Key Trustee KMS binary using packages or parcels**.
2. Select the KEYTRUSTEE_SERVER parcel to install Key Trustee Server, or select **None** if you need to install Key Trustee Server manually using packages. If you do not see a parcel available, click **More Options** and add the repository URL to the **Remote Parcel Repository URLs** list. After selecting a parcel, click **Continue**.
3. After the KEYTRUSTEE_SERVER parcel is successfully downloaded, distributed, unpacked, and activated, click **Finish** to complete this step and return to the main page of the wizard.

5. Install Parcel for Key Trustee KMS

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

This step installs the Key Trustee KMS parcel. If you are using packages, skip this step and see Installing Key Trustee KMS Using Packages for instructions. After installing Key Trustee KMS using packages, continue to 6. Add a Key Trustee Server Service on page 242.

To complete this step for parcel-based installations:

1. Click **Install Parcel for Key Trustee KMS**.
2. Select the KEYTRUSTEE parcel to install Key Trustee KMS. If you do not see a parcel available, click **More Options** and add the repository URL to the **Remote Parcel Repository URLs** list. After selecting a parcel, click **Continue**.
3. After the KEYTRUSTEE parcel is successfully downloaded, distributed, unpacked, and activated, click **Finish** to complete this step and return to the main page of the wizard.

6. Add a Key Trustee Server Service

Minimum Required Role: Key Administrator (also provided by **Full Administrator**)

This step adds the **Key Trustee Server** service to Cloudera Manager. To complete this step:

1. Click **Add a Key Trustee Server Service**.
2. Click **Continue**.
3. On the **Customize Role Assignments for Key Trustee Server** page, select the hosts for the **Active Key Trustee Server** and **Passive Key Trustee Server** roles. Make sure that the selected hosts are not used for other services (see Resource Planning for Data at Rest Encryption on page 233 for more information), and click **Continue**.
4. The **Entropy Considerations** page provides commands to install the `rng-tools` package to increase available entropy for cryptographic operations. For more information, see Entropy Requirements on page 229. After completing these commands, click **Continue**.

5. The **Synchronize Active and Passive Key Trustee Server Private Keys** page provides instructions for generating and copying the Active Key Trustee Server private key to the Passive Key Trustee Server. Cloudera recommends following security best practices and transferring the private key using offline media, such as a removable USB drive. For convenience (for example, in a development or testing environment where maximum security is not required), you can copy the private key over the network using the provided `rsync` command.

   After you have synchronized the private keys, run the `ktadmin init` command on the Passive Key Trustee Server as described in the wizard. After the initialization is complete, check the box to indicate you have synchronized the keys and click **Continue** in the wizard.

6. The **Setup TLS for Key Trustee Server** page provides instructions on replacing the auto-generated self-signed certificate with a production certificate from a trusted Certificate Authority (CA). For more information, see Managing Key Trustee Server Certificates on page 298. Click **Continue** to view and modify the default certificate settings.
7. On the **Review Changes** page, you can view and modify the following settings:

   - **Database Storage Directory** (`db_root`)

     Default value: `/var/lib/keytrustee/db`

     The directory on the local filesystem where the Key Trustee Server database is stored. Modify this value to store the database in a different directory.

   - **Active Key Trustee Server TLS/SSL Server Private Key File (PEM Format)** (`ssl.privatekey.location`)

     Default value: `/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-pk.pem`

     The path to the Active Key Trustee Server TLS certificate private key. Accept the default setting to use the auto-generated private key. If you have a CA-signed certificate, change this path to the CA-signed certificate private key file. This file must be in PEM format.

   - **Active Key Trustee Server TLS/SSL Server Certificate File (PEM Format)** (`ssl.cert.location`)

     Default value: `/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.pem`

     The path to the Active Key Trustee Server TLS certificate. Accept the default setting to use the auto-generated self-signed certificate. If you have a CA-signed certificate, change this to the path to the CA-signed certificate. This file must be in PEM format.

   - **Active Key Trustee Server TLS/SSL Server CA Certificate (PEM Format)** (`ssl.cacert.location`)

     Default value: (none)

     The path to the file containing the CA certificate and any intermediate certificates used to sign the Active Key Trustee Server certificate. If you have a CA-signed certificate, set this value to the path to the CA certificate or certificate chain file. This file must be in PEM format.

   - **Active Key Trustee Server TLS/SSL Private Key Password** (`ssl.privatekey.password`)

     Default value: (none)

     The password for the Active Key Trustee Server private key file. Leave this blank if the file is not password-protected.

   - **Passive Key Trustee Server TLS/SSL Server Private Key File (PEM Format)** (`ssl.privatekey.location`)

     Default value: `/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-pk.pem`

     The path to the Passive Key Trustee Server TLS certificate private key. Accept the default setting to use the auto-generated private key. If you have a CA-signed certificate, change this path to the CA-signed certificate private key file. This file must be in PEM format.

   - **Passive Key Trustee Server TLS/SSL Server Certificate File (PEM Format)** (`ssl.cert.location`)

     Default value: `/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.pem`

The path to the Passive Key Trustee Server TLS certificate. Accept the default setting to use the auto-generated self-signed certificate. If you have a CA-signed certificate, change this to the path to the CA-signed certificate. This file must be in PEM format.

- **Passive Key Trustee Server TLS/SSL Server CA Certificate (PEM Format)** (`ssl.cacert.location`)

  Default value: (none)

  The path to the file containing the CA certificate and any intermediate certificates used to sign the Passive Key Trustee Server certificate. If you have a CA-signed certificate, set this value to the path to the CA certificate or certificate chain file. This file must be in PEM format.

- **Passive Key Trustee Server TLS/SSL Private Key Password** (`ssl.privatekey.password`)

  Default value: (none)

  The password for the Passive Key Trustee Server private key file. Leave this blank if the file is not password-protected.

After reviewing the settings and making any changes, click **Continue**.

8. After all commands complete successfully, click **Continue**. If the **Generate Key Trustee Server Keyring** appears stuck, make sure that the Key Trustee Server host has enough entropy. See Entropy Requirements on page 229 for more information.
9. Click **Finish** to complete this step and return to the main page of the wizard.

For parcel-based Key Trustee Server releases 5.8 and higher, Cloudera Manager automatically backs up Key Trustee Server (using the `ktbackup.sh` script) after adding the Key Trustee Server service. It also schedules automatic backups using `cron`. For package-based installations, you must manually back up Key Trustee Server and configure a `cron` job.

Cloudera Manager configures `cron` to run the backup script hourly. The latest 10 backups are retained in `/var/lib/keytrustee` in cleartext. For information about using the backup script and configuring the `cron` job (including how to encrypt backups), see Backing Up Key Trustee Server and Key Trustee KMS Using the ktbackup.sh Script on page 283.

7. Add a Key Trustee KMS Service

Minimum Required Role: Key Administrator (also provided by **Full Administrator**)

This step adds a Key Trustee KMS service to the cluster. The Key Trustee KMS service is required to enable HDFS encryption to use Key Trustee Server for cryptographic key management. Key Trustee KMS high availability uses ZooKeeper to automatically configure load balancing. If you do not have a ZooKeeper service in your cluster, add one using the instructions in Adding a Service.

To complete this step:

1. Click **Add a Key Trustee KMS Service**.
2. Select an existing Key Trustee Server pair or specify an external Key Trustee Server pair. If you have an existing Key Trustee Server pair outside of Cloudera Manager control, select the **External Key Trustee Server** option and specify the fully-qualified domain names (FQDNs) of the Key Trustee Server pair. Click **Continue**.
3. Select cluster hosts for the Key Trustee KMS service. For production environments, select at least two hosts for high availability. If you proceed with only one host, you can enable high availability later. See Enabling Key Trustee KMS High Availability for more information.

   Make sure that the selected hosts are not used for other services (see Resource Planning for Data at Rest Encryption on page 233 for more information), and click **Continue**.
4. The **Entropy Considerations** page provides commands to install the `rng-tools` package to increase available entropy for cryptographic operations. For more information, see Entropy Requirements on page 229. After completing these commands, click **Continue**.

5. The **Setup Organization and Auth Secret** page generates the necessary commands to create an organization in Key Trustee Server. An organization is required to be able to register the Key Trustee KMS with Key Trustee Server. See Managing Key Trustee Server Organizations on page 295 for more information.

   Enter an organization name and click **Generate Instruction**. Run the displayed commands to generate an organization and obtain the `auth_secret` value for the organization. Enter the secret in the **auth_secret** field and click **Continue**.

6. The **Setup Access Control List (ACL)** page allows you to generate ACLs for the Key Trustee KMS or to provide your own ACLs. To generate the recommended ACLS, enter the username and group responsible for managing cryptographic keys and click **Generate ACLs**. To specify your own ACLs, select the **Use Your Own kms-acls.xml File** option and enter the ACLs. For more information on the KMS Access Control List, see Configuring KMS Access Control Lists on page 267.

   After generating or specifying the ACL, click **Continue**.

7. The **Setup TLS for Key Trustee KMS** page provides high-level instructions for configuring TLS communication between the Key Trustee KMS and the Key Trustee Server, as well as between the EDH cluster and the Key Trustee KMS. See Configuring TLS/SSL for the KMS on page 265 for more information.

   Click **Continue**.

8. The **Review Changes** page lists all of the settings configured in this step. Click the icon next to any setting for information about that setting. Review the settings and click **Continue**.
9. After the **First Run** commands have successfully completed, click **Continue**.
10. The **Synchronize Private Keys and HDFS Dependency** page provides instructions for copying the private key from one Key Management Server Proxy role to all other roles.

   > **Warning:** It is *very important* that you perform this step. Failure to do so leaves Key Trustee KMS in a state where keys are intermittently inaccessible, depending on which Key Trustee KMS host a client interacts with, because cryptographic key material encrypted by one Key Trustee KMS host cannot be decrypted by another. If you are already running multiple Key Trustee KMS hosts with different private keys, immediately back up all Key Trustee KMS hosts, and contact Cloudera Support for assistance correcting the issue.
   >
   > To determine whether the Key Trustee KMS private keys are different, compare the MD5 hash of the private keys. On each Key Trustee KMS host, run the following command:
   >
   > ```
   > $ md5sum /var/lib/kms-keytrustee/keytrustee/.keytrustee/secring.gpg
   > ```
   >
   > If the outputs are different, contact Cloudera Support for assistance. Do not attempt to synchronize existing keys. If you overwrite the private key and do not have a backup, any keys encrypted by that private key are permanently inaccessible, and any data encrypted by those keys is permanently irretrievable. If you are configuring Key Trustee KMS high availability for the first time, continue synchronizing the private keys.

   Cloudera recommends following security best practices and transferring the private key using offline media, such as a removable USB drive. For convenience (for example, in a development or testing environment where maximum security is not required), you can copy the private key over the network using the provided `rsync` command.

   After you have synchronized the private keys, check the box to indicate you have done so and click **Continue**.

11. After the Key Trustee KMS service starts, click **Finish** to complete this step and return to the main page of the wizard.

For parcel-based Key Trustee KMS releases 5.8 and higher, Cloudera Manager automatically backs up Key Trustee KMS (using the `ktbackup.sh` script) after adding the Key Trustee KMS service. It does not schedule automatic backups using `cron`. For package-based installations, you must manually back up Key Trustee Server and configure a `cron` job.

The backup is stored in `/var/lib/kms-keytrustee` in cleartext. For more information about using the backup script and configuring the `cron` job (including how to encrypt backups), see Backing Up Key Trustee Server and Key Trustee KMS Using the ktbackup.sh Script on page 283.

8. Restart stale services and redeploy client configuration

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

This step restarts all services which were modified while enabling HDFS encryption. To complete this step:

1. Click **Restart stale services and redeploy client configuration**.
2. Click **Restart Stale Services**.
3. Ensure that **Re-deploy client configuration** is checked, and click **Restart Now**.
4. After all commands have completed, click **Finish**.

9. Validate Data Encryption

Minimum Required Role: Key Administrator *or* **Cluster Administrator** (also provided by Full Administrator)

This step launches a tutorial with instructions on creating an encryption zone and putting data into it to verify that HDFS encryption is enabled and working.

Enabling HDFS Encryption Using Navigator HSM KMS Backed by Thales HSM

1. Enable Kerberos

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

For more information about enabling Kerberos, see Enabling Kerberos Authentication Using the Wizard on page 50.

2. Enable TLS/SSL

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

For more information about enabling TLS, see Configuring Cloudera Manager Clusters for TLS/SSL on page 183.

3. Install the Thales HSM Client

Before installing the Navigator HSM KMS backed by Thales HSM, you *must* install the Thales HSM client on the host. Attempts to install the HSM KMS service before installing the Thales HSM client will fail.

For details about how to install the Thales HSM client, refer to the Thales HSM product documentation.

4. Install Key Trustee KMS binary using parcels

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

This step completes automatically when you download the parcel. If the step is incomplete for any reason (such as the wizard being interrupted or a failure installing the parcel), complete it manually:

1. Click **Install Key Trustee KMS binary using parcels**.
2. Select the KEYTRUSTEE parcel to install Key Trustee KMS, or select **None** if you need to install Key Trustee KMS manually using packages. If you do not see a parcel available, click **More Options** and add the repository URL to the **Remote Parcel Repository URLs** list. After selecting a parcel, click **Continue**.
3. After the KEYTRUSTEE parcel is successfully downloaded, distributed, unpacked, and activated, click **Finish** to complete this step and return to the main page of the wizard.

5. Add the HSM KMS backed by Thales Service

1. Click **Add Navigator HSM KMS Services backed by Thales HSM**.

2. In the Thales HSM KMS Proxy field, select the hosts to which you want to assign a new or existing role. Click **OK**, and then click **Continue**.
3. To set up the ACL for the cluster, specify a comma-separated list of users and groups, and then click **Generate ACLs**. Click **Continue**.
4. Click **Continue**.
5. Review your selections and specify the:
   - Thales HSM Password

     Contact your HSM administrator for the Thales HSM password.
   - Keystore Password

     > **Important:** Specify a unique password to protect the KMS metadata. *Safely store this password for future use*. In the event that the service is deleted, you *must* provide this password to re-add the service with the same metastore.

   > **Note:** It is a good practice to review the Thales HSM Server port at this point, because it could have changed during configuration.

   Then click **Continue**.
6. Upon notification that you have successfully added the Thales KMS Service, click **Continue** and **Finish**.

6. Restart stale services and redeploy client configuration

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

This step restarts all services that were modified while enabling HDFS encryption. To complete this step:

1. Click **Restart stale services and redeploy client configuration**.
2. Click **Restart Stale Services**.
3. Ensure that **Re-deploy client configuration** is checked, and click **Restart Now**.
4. After all commands have completed, click **Finish**.

7. Validate Data Encryption

Minimum Required Role: Key Administrator *or* **Cluster Administrator** (also provided by Full Administrator)

This step launches a Validate Data Encryption tutorial with instructions describing how to create an encryption zone and place data into it to verify that HDFS encryption is enabled and working.

Enabling HDFS Encryption Using Navigator HSM KMS Backed by Luna HSM

1. Enable Kerberos

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

For more information about enabling Kerberos, see Enabling Kerberos Authentication Using the Wizard on page 50.

2. Enable TLS/SSL

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

For more information about enabling TLS, see Configuring Cloudera Manager Clusters for TLS/SSL on page 183.

3. Install Luna HSM Client

Before installing the Navigator HSM KMS backed by Luna HSM, you must install the Luna HSM client on the host. Attempts to install the Navigator HSM KMS backed by Luna HSM before installing the Luna HSM client will fail.

For details about how to install the Luna HSM client, refer to the Luna HSM product documentation.

4. Install Parcel for Cloudera Key Providers

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

This step completed automatically when you downloaded the parcel. If the step is incomplete for any reason (such as the wizard being interrupted or a failure installing the parcel), complete it manually:

1. Click **Install Key Trustee KMS binary using parcels**.
2. Select the KEYTRUSTEE parcel to install Key Trustee KMS, or select **None** if you need to install Key Trustee KMS manually using packages. If you do not see a parcel available, click **More Options** and add the repository URL to the **Remote Parcel Repository URLs** list. After selecting a parcel, click **Continue**.
3. After the KEYTRUSTEE parcel is successfully downloaded, distributed, unpacked, and activated, click **Finish** to complete this step and return to the main page of the wizard.

5. Add the Navigator HSM KMS backed by SafeNet Luna HSM

1. Click **Add Navigator HSM KMS backed by Safenet Luna HSM**.
2. In the **Luna HSM-backed KMS Proxy** field, select the hosts to which you want to assign a new or existing role. Click **OK**, and then click **Continue**.
3. To set up the ACL for the cluster, specify a comma-separated list of users and groups, and then click **Generate ACLs**. Click **Continue**.
4. Click **Continue**.
5. Review your selections and specify the:
   - Luna HSM Password

     Contact your HSM administrator for the Luna HSM Partition password.
   - Keystore Password

     > **Important:** Specify a unique password to protect the KMS metadata. *Safely store this password for future use*. In the event that the service is deleted, you *must* provide this password to re-add the service with the same metastore.
   - Luna HSM Server Slot

     Identification number of the Luna HSM Server slot/device to use. If you do not know what value(s) to enter here, see the Luna product documentation for instructions on configuring your Luna HSM. Alternatively, run the `/usr/safenet/lunaclient/bin/vtl verify` command on the Luna HSM client host to view the slot value.

   Then click **Continue**.
6. Upon notification that you have successfully added the Navigator Safenet Luna KMS Service, click **Continue** and **Finish**.

6. Restart stale services and redeploy client configuration

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

This step restarts all services that were modified while enabling HDFS encryption. To complete this step:

1. Click **Restart stale services and redeploy client configuration**.
2. Click **Restart Stale Services**.
3. Ensure that **Re-deploy client configuration** and **HDFS-1** are checked, and click **Restart Now**.
4. After all commands have completed, click **Finish**.

7. Validate Data Encryption

Minimum Required Role: Key Administrator *or* **Cluster Administrator** (also provided by Full Administrator)

This step launches a Validate Data Encryption tutorial with instructions describing how to create an encryption zone and place data into it to verify that HDFS encryption is enabled and working.

Enabling HDFS Encryption Using a Java KeyStore

> **Note:** Cloudera strongly recommends using Cloudera Navigator Key Trustee Server as the root of trust for production environments. The file-based Java KeyStore root of trust is insufficient to provide the security, scalability, and manageability required by most production systems.

After selecting **A file-based password-protected Java KeyStore** as the root of trust, the following steps are displayed:

1. Enable Kerberos

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

For more information on enabling Kerberos, see Enabling Kerberos Authentication Using the Wizard on page 50.

2. Enable TLS/SSL

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

For more information on enabling TLS, see Configuring Cloudera Manager Clusters for TLS/SSL on page 183.

3. Add a Java KeyStore KMS Service

Minimum Required Role: Key Administrator (also provided by **Full Administrator**)

This step adds the Java KeyStore KMS service to the cluster. The Java KeyStore KMS service uses a password-protected Java KeyStore for cryptographic key management. To complete this step:

1. Click **Add a Java KeyStore KMS Service**.
2. Select a cluster host for the Java KeyStore KMS service. Click **Continue**.
3. The **Setup TLS for Java KeyStore KMS** page provides high-level instructions for configuring TLS communication between the EDH cluster and the Java KeyStore KMS. See Configuring TLS/SSL for the KMS on page 265 for more information.

   Click **Continue**.
4. The **Review Changes** page lists the Java KeyStore settings. Click the icon next to any setting for information about that setting. Enter the location and password for the Java KeyStore and click **Continue**.
5. Click **Continue** to automatically configure the HDFS service to depend on the Java KeyStore KMS service.
6. Click **Finish** to complete this step and return to the main page of the wizard.

4. Restart stale services and redeploy client configuration

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

This step restarts all services which were modified while enabling HDFS encryption. To complete this step:

1. Click **Restart stale services and redeploy client configuration**.
2. Click **Restart Stale Services**.
3. Ensure that **Re-deploy client configuration** is checked, and click **Restart Now**.
4. After all commands have completed, click **Finish**.

5. Validate Data Encryption

Minimum Required Role: Key Administrator *or* **Cluster Administrator** (also provided by Full Administrator)

This step launches a tutorial with instructions on creating an encryption zone and putting data into it to verify that HDFS encryption is enabled and working.

Hints and Tips

This section includes hints and tips that can help simplify the HSM KMS installation when using the HDFS Encryption Wizard.

Limit the Number of ZooKeeper DEBUG Messages

When setting the KMS log level to DEBUG, there can be a lot of ZooKeeper DEBUG messages that clutter the log. To prevent this, in the **LUNA HSM-backed KMS Proxy Logging Advanced Configuration Snippet (Safety Valve)** field, enter:

```
log4j.category.org.apache.zookeeper=INFO
```

Limit Encryption Zone Timeouts

When creating encryption zones, there can be client timeouts due to the time it takes to fill the encrypted data encryption key (EDEK) cache. To avoid this, adjust the low watermark threshold settings as follows.

On the server side, in the field **HSM KMS Proxy Advanced Configuration Snippet (Safety Valve) for kms-site.xml**:

```
<property>
   <name>hadoop.security.kms.encrypted.key.cache.low.watermark</name>
   <value>.03</value>
</property>
```

On the client side, in the field **HDFS Cluster-wide Advanced Configuration Snippet (Safety Valve) for core-site.xml**:

```
<property>
   <name>hadoop.security.kms.client.encrypted.key.cache.low-watermark</name>
   <value>.02</value>
</property>
```

Increase KMS Client Timeout Value

Due to potential latency during installation, it is recommended that you increase the KMS client timeout value.

Change from the default of 60 seconds to a value between 100 and 120 seconds in the field **HDFS Cluster-wide Advanced Configuration Snippet (Safety Valve) for core-site.xml**:

```
<property>
   <name>hadoop.security.kms.client.timeout</name>
   <value>110</value>
</property>
```

Managing Encryption Keys and Zones

Interacting with the KMS and creating encryption zones requires the use of two new CLI commands: `hadoop key` and `hdfs crypto`. The following sections will help you get started with creating encryption keys and setting up encryption zones.

Before continuing, make sure that your KMS ACLs have been set up according to best practices. For more information, see Configuring KMS Access Control Lists on page 267.

Validating Hadoop Key Operations

> **Warning:** If you are using or plan to use Cloudera Navigator Key HSM in conjunction with Cloudera Navigator Key Trustee Server, ensure that key names begin with alphanumeric characters and do not use special characters other than hyphen (`-`), period (`.`), or underscore (`_`). Using other special characters can prevent you from migrating your keys to an HSM. See Integrating Key HSM with Key Trustee Server on page 306 for more information.

Use `hadoop key create` to create a test key, and then use `hadoop key list` to retrieve the key list:

```
$ sudo -u <key_admin> hadoop key create keytrustee_test
$ hadoop key list
```

Creating Encryption Zones

> **Important:** Cloudera does not currently support configuring the root directory as an encryption zone. Nested encryption zones are also not supported.

> **Important:** The Java Keystore KMS default Truststore (for example, `org.apache.hadoop.cryto.key.JavaKeyStoreProvider`) does not support uppercase key names.

Once a KMS has been set up and the NameNode and HDFS clients have been correctly configured, use the hadoop key and `hdfs crypto` command-line tools to create encryption keys and set up new encryption zones.

- Create an encryption key for your zone as the application user that will be using the key. For example, if you are creating an encryption zone for HBase, create the key as the `hbase` user as follows:

```
$ sudo -u hbase hadoop key create <key_name>
```

- Create a new empty directory and make it an encryption zone using the key created above.

```
$ sudo -u hdfs hadoop fs -mkdir /encryption_zone
$ sudo -u hdfs hdfs crypto -createZone -keyName <key_name> -path /encryption_zone
```

You can verify creation of the new encryption zone by running the `-listZones` command. You should see the encryption zone along with its key listed as follows:

```
$ sudo -u hdfs hdfs crypto -listZones
/encryption_zone    <key_name>
```

> **Warning:** Do not delete an encryption key as long as it is still in use for an encryption zone. This results in loss of access to data in that zone.

For more information and recommendations on creating encryption zones for each CDH component, see Configuring CDH Services for HDFS Encryption on page 276.

Adding Files to an Encryption Zone

You can add files to an encryption zone by copying them to the encryption zone using `distcp`. For example:

```
sudo -u hdfs hadoop distcp /user/dir /encryption_zone
```

> **Important:** Starting with CDH 5.7.1, you can delete files or directories that are part of an HDFS encryption zone. For CDH 5.7.0 and lower, you will need to manually configure HDFS trash to allow deletions. For details on how to configure trash in HDFS, see Trash Behavior with HDFS Transparent Encryption Enabled.

DistCp Considerations

A common use case for DistCp is to replicate data between clusters for backup and disaster recovery purposes. This is typically performed by the cluster administrator, who is an HDFS superuser. To retain this workflow when using HDFS encryption, a new virtual path prefix has been introduced, `/.reserved/raw/`, that gives superusers direct access to the underlying block data in the filesystem. This allows superusers to `distcp` data without requiring access to encryption keys, and avoids the overhead of decrypting and re-encrypting data. It also means the source and destination data will be byte-for-byte identical, which would not have been true if the data was being re-encrypted with a new EDEK.

> **Warning:**
>
> When using `/.reserved/raw/` to `distcp` encrypted data, make sure you preserve extended attributes with the `-px` flag. This is because encrypted attributes such as the EDEK are exposed through extended attributes and *must* be preserved to be able to decrypt the file.
>
> This means that if the `distcp` is initiated at or above the encryption zone root, it will automatically create a new encryption zone at the destination if it does not already exist. Hence, Cloudera recommends you first create identical encryption zones on the destination cluster to avoid any potential mishaps.

Copying between encrypted and unencrypted locations

By default, `distcp` compares checksums provided by the filesystem to verify that data was successfully copied to the destination. When copying between an unencrypted and encrypted location, the filesystem checksums will not match since the underlying block data is different.

In this case, you can specify the `-skipcrccheck` and `-update` flags to avoid verifying checksums. When you use -skipcrccheck, `distcp` checks the file integrity by performing a file size comparison, right after the copy completes for each file.

Deleting Encryption Zones

To remove an encryption zone, delete the encrypted directory:

> **Warning:** This command deletes the entire directory and all of its contents. Ensure that the data is no longer needed before running this command.

```
$ sudo -u hdfs hadoop fs -rm -r -skipTrash /encryption_zone
```

> **Important:** The Key Trustee KMS does not directly execute a key deletion (for example, it may perform a soft delete instead, or delay the actual deletion to prevent mistakes). In these cases, errors may occur when creating or deleting a key using the same name after it has already been deleted.

Backing Up Encryption Keys

> **Warning:** It is *very important* that you regularly back up your encryption keys. Failure to do so can result in irretrievable loss of encrypted data.

If you are using the Java KeyStore KMS, make sure you regularly back up the Java KeyStore that stores the encryption keys. If you are using the Key Trustee KMS and Key Trustee Server, see Backing Up and Restoring Key Trustee Server and Clients on page 283 for instructions on backing up Key Trustee Server and Key Trustee KMS.

Rolling Encryption Keys

Before attempting to roll an encryption key (also known as an encryption zone key, or EZ key), familiarize yourself with the concepts described in Cloudera Navigator Data Encryption Overview, and HDFS Transparent Encryption, as the material in these sections presumes you are familiar with the fundamentals of HDFS transparent encryption and Cloudera data at rest encryption.

When you roll an EZ key, you are essentially creating a new version of the key (`ezKeyVersionName`). Rolling EZ keys regularly helps enterprises minimize the risk of key exposure. If a malicious attacker were to obtain the EZ key and decrypt encrypted data encryption keys (EDEKs) into DEKs, they could gain the ability to decrypt HDFS files. Rolling an EZ key ensures that all DEKs for newly-created files will be encrypted with the new version of the EZ key. The older EZ key version that the attacker obtained cannot decrypt these EDEKs. You may want to roll the encryption key periodically, as part of your security policy or when an external security compromise is detected.

To roll an EZ key:

1. (***Optional***) Before rolling any keys, log in as HDFS Superuser and verify/identify the encryption zones to which the current key applies. This operation also helps clarify the relationship between the EZ key and encryption zones, and, if necessary, makes it easier to identify more important, high priority zones:

```
$ hdfs crypto –listZones
/ez key1
/ez2 key2
/user key1
```

   The first column identifies the encryption zone paths; the second column identifies the encryption key name.

2. (***Optional***) You can verify that the files inside an encryption zone are encrypted using the `hdfs crypto -getFileEncryptionInfo` command. Note the EZ key version name and value, which you can use for comparison and verification after rolling the EZ key.

```
$ hdfs crypto –getFileEncryptionInfo –path /ez/f
{cipherSuite: {name: AES/CTR/NoPadding, algorithmBlockSize: 16}. cryptoProtocolVersion:
CryptoProtocolVersion{description=’Encryption zones’, version=2, unknownValue=null},
edek: 373c0c2e919c27e58c1c343f54233cbd,
iv: d129c913c8a34cde6371ec95edfb7337, keyName: key1, ezKeyVersionName:
7mbvopZ0Weuvs0XtTkpGw3G92KuWc4e4xcTXl0bXCpF}
```

   Log off as HDFS Superuser.

3. Log in as Key Administrator. Because keys can be rolled, a key can have multiple key versions, where each key version has its own key material (the actual secret bytes used during DEK encryption and EDEK decryption). You can fetch an encryption key by either its key name, returning the latest version of the key, or by a specific key version.

   Roll the encryption key (previously identified/confirmed by the HDFS Superuser in step 1. Here, the `<key name>` is `key1`:

```
$ hadoop key roll key1
```

   This operation contacts the KMS and rolls the keys there. Note that this can take a considerable amount of time, depending on the number of key versions residing in the KMS.

```
Rolling key version from KeyProvider:
org.apache.hadoop.crypto.key.kms.LoadBalancingKMSClientProvider@5ea434c8
for keyName: key1
key1 has been successfully rolled.
org.apache.hadoop.crypto.key.kms.LoadBalancingKMSClientProvider@5ea434c8 has been updated.
```

4. (***Optional***) Log out as Key Administrator, and log in as HDFS Superuser. Verify that new files in the encryption zone have a new EZ key version.

> **Note:** For performance reasons, the NameNode caches EDEKs, so after rolling an encryption key, you may not be able to see the new version encryption key immediately, or at least until after the EDEK cache is consumed. Of course, the file decryption and encryption still works with these EDEKs. If you require that all files' DEKs in an encryption zone are encrypted using the latest version encryption key, please re-encrypt the EDEKs.

```
$ hdfs crypto –getFileEncryptionInfo –path /ez/new_file
{cipherSuite: {name: AES/CTR/NoPadding, algorithmBlockSize: 16}. cryptoProtocolVersion:
CryptoProtocolVersion{description=’Encryption zones’, version=2, unknownValue=null},
edek: 9aa13ea4a700f96287cfe1349f6ff4f2,
iv: 465c878ad9325e42fa460d2a22d12a72, keyName: key1, ezKeyVersionName:
4tuvorJ6Feeqk8WiCfdDs9K32KuEj7g2ydCAv0gNQbY}
```

Alternatively, you can use KMS Rest API to view key metadata and key versions. Elements appearing in brackets should be replaced with your actual values. So in this case, before rolling a key, you can view the key metadata and versions as follows:

```
$ curl -k --negotiate -u: "https://<KMS_HOSTNAME>:16000/kms/v1/key/<key-name>/_metadata"
{
  "name" : "<key-name>",
  "cipher" : "<cipher>",
  "length" : <length>,
  "description" : "<decription>",
  "created" : <millis-epoc>,
  "versions" : <versions> (For example, 1)
}

$ curl -k  --negotiate -u:
"https://<KMS_HOSTNAME>:16000/kms/v1/key/<key-name>/_currentversion"
{
  "material" : "<material>",
  "name" : "<key-name>",
  "versionName" : "<versionName>" (For example, version 1)
}
```

Roll the key and compare the results:

```
$ hadoop key roll key1

Rolling key version from KeyProvider:
KMSClientProvider[https://<KMS_HOSTNAME>:16000/kms/v1/]

  for key name: <key-name>

key1 has been successfully rolled.

KMSClientProvider[https://<KMS_HOSTNAME>/kms/v1/] has been updated.

$ curl -k  --negotiate -u:
"https://<KMS_HOSTNAME>:16000/kms/v1/key/<key-name>/_currentversion"

{
  "material" : "<material>", (New material)
  "name" : "<key-name>",
  "versionName" : "<versionName>" (New version name. For example, version 2)
}

$ curl -k --negotiate -u: "https://<KMS_HOSTNAME>:16000/kms/v1/key/<key-name>/_metadata"
{
  "name" : "<key-name>",
  "cipher" : "<cipher>",
  "length" : <length>,
  "description" : "<decription>",
  "created" : <millis-epoc>,
```

```
  "versions" : <versions> (For example, version 2)
}
```

Re-encrypting Encrypted Data Encryption Keys (EDEKs)

Before attempting to re-encrypt an EDEK, familiarize yourself with the concepts described in Cloudera Navigator Data Encryption Overview, and HDFS Transparent Encryption, as the material in this section presumes you are familiar with the fundamentals of HDFS transparent encryption and Cloudera data at rest encryption.

When you re-encrypt an EDEK, you are essentially decrypting the original EDEK created by the DEK, and then re-encrypting it using the new (rolled) version of the EZ key (see Rolling Encryption Keys on page 253). The file's metadata, which is stored in the NameNode, is then updated with this new EDEK. Re-encryption does not impact the data in the HDFS files or the DEK--the same DEK is still used to decrypt the file, so re-encryption is essentially transparent.

Benefits and Capabilities

In addition to minimizing security risks, re-encrypting the EDEK offers the following capabilities and benefits:

- Re-encrypting EDEKs does *not* require that the user explicitly re-encrypt HDFS files.
- In cases where there are several zones using the same key, the Key Administrator has the option of selecting which zone's EDEKs are re-encrypted first.
- The HDFS Superuser can also monitor and cancel re-encryption operations.
- Re-encryption is restarted automatically in cases where you have a NameNode failure during the re-encryption operation.

Prerequisites and Assumptions

Before attempting to re-encrypt an EDEK, familiarize yourself with the concepts and rules described in Managing Encryption Keys and Zones on page 250.

- It is recommended that you perform EDEK re-encryption at the same time that you perform regular cluster maintenance because the operation can adversely impact CPU resources on the NameNode.
- The following account credentials and settings must exist before you can re-encrypt EDEKs: CDH 5.13.0 with KT KMS 5.13.0.
- In Cloudera Manager, review the cluster's NameNode status, which must be in "Good Health". If the cluster NameNode does not have a status of "Good Health", then do not proceed with the re-encryption of the EDEK. In the Cloudera Manager WebUI menu, you can verify the status for the cluster NameNode, which must *not* be in Safe mode (in other words, the WebUI should indicate "Safemode is off").

Running the re-encryption command without successfully verifying the preceding items will result in failures with errors.

Limitations

This section identifies limitations associated with the re-encryption of EDEKs.

EDEK re-encryption doesn't change EDEKs on snapshots, due to the immutable nature HDFS snapshots. Thus, you should be aware that after EZ key exposure, the Key Administrator must delete snapshots.

Re-encrypting an EDEK

This scenario operates on the assumption that an encryption zone has already been set up for this cluster. For more details about creating an encryption zone, see Creating Encryption Zones on page 251.

1. Navigate to the cluster in which you will be rolling keys and re-encrypting the EDEK.
2. Log in as HDFS Superuser.
3. (***Optional***) To view all of the options for the `hdfs crypto` command:

```
$ hdfs crypto
[-createZone –keyName <keyName> -path <path>]
```

```
[-listZones]
[-provisionTrash -path <path>]
[-getFileEncryptionInfo -path <path>]
[-reencryptZone <action> -path <zone>]
[-listReencryptionStatus]
[-help <command-name>]
```

4. Before rolling any keys, verify/identify the encryption zones to which the current key applies. This operation also helps clarify the relationship between the EZ key and encryption zones, and, if necessary, makes it easier to identify more important, high priority zones:

```
$ hdfs crypto –listZones
/ez     key1
```

The first column identifies the encryption zone path (`/ez`); the second column identifies the encryption key name (`key1`).

5. Exit from the HDFS Superuser account and log in as Key Administrator.
6. Roll the encryption key (previously identified/confirmed by the HDFS Superuser in step 4). Here, the `<key name>` is `key1`:

```
$ hadoop key roll key1
```

This operation contacts the KMS and rolls the keys. Note that this can take a considerable amount of time, depending on the number of key versions.

```
Rolling key version from KeyProvider:
org.apache.hadoop.crypto.key.kms.LoadBalancingKMSClientProvider@5ea434c8
for keyName: key1
key1 has been successfully rolled.
org.apache.hadoop.crypto.key.kms.LoadBalancingKMSClientProvider@5ea434c8 has been updated.
```

7. Log out as Key Administrator, and log in as HDFS Superuser.
8. (***Optional***) Before performing the re-encryption, you can verify the status of the current key version being used (keyName). Then, after re-encrypting, you can confirm that the EZ key version (`ezKeyVersionName`) and EDEK have changed:

```
$ hdfs crypto –getFileEncryptionInfo –path /ez/f
{cipherSuite: {name: AES/CTR/NoPadding, algorithmBlockSize: 16}. cryptoProtocolVersion:
CryptoProtocolVersion{description='Encrytpion zones', version=2, unknownValue=null},
edek: 9aa13ea4a700f96287cfe1349f6ff4f2,
iv: d129c913c8a34cde6371ec95edfb7337, keyName: key1, ezKeyVersionName:
7mbvopZ0Weuvs0XtTkpGw3G92KuWc4e4xcTXl0bXCpF}
```

9. After the EZ key has been rolled successfully, re-encrypt the EDEK by running the re-encryption command on the encryption zone:

```
$ hdfs crypto –reencryptZone –start –path /ez
```

The following information appears when the submission is complete. At this point, the NameNode is processing and re-encrypting all of the EDEKs under the `/ez` directory.

```
re-encrypt command successfully submitted for zone: /ez action:  START:
```

Depending on the number of files, the re-encryption operation can take a long time. Re-encrypting a 1M EDEK file typically takes between 2-6 minutes, depending on the NameNode hardware.To check the status of the re-encryption for the zone:

```
$ hdfs crypto –listReencryptionStatus
```

Table 20: Re-encryption Status Column Data

Column Name	Description	Sample Data
ZoneName	The encryption zone name	/ez
Status	• Submitted: the command is received, but not yet being processed by the NameNode. • Processing: the zone is being processed by the NameNode. • Completed: the NameNode has finished processing the zone, and every file in the zone has been re-encrypted.	Completed
EZKey Version Name	The encryption zone key version name, which used for re-encryption comparison. After re-encryption is complete, all files in the encryption zone are guaranteed to have an EDEK whose encryption zone key version is at least equal to this version.	ZMIHfRoGKeXXgf0QzCX8q16NczIw2sq0rWRTOHS3YjCz
Submission Time	The time at which the re-encryptiom operation commenced.	2017-09-07 10:01:09,262-0700
Is Canceled?	True: the encryption operation has been canceled. False: the encryption operation has not been canceled.	False
Completion Time	The time at which the re-encryption operation completed.	2017-09-07 10:01:10,441-0700
Number of files re-encrypted	The number that appears in this column reflects *only* the files whose EDEKs have been updated. If a file is created after the key is rolled, then it will already have an EDEK that has been encrypted by the new key version, so the re-encryption operation will skip that file. In other words, it's possible for a "Completed" re-encryption to reflect a number of re-encrypted files that is less than the number of files actually in the encryption zone. **Note:** In cases when you re-encrypt an EZ key that has already been re-encrypted and there are no new files, the number of files re-encrypted will be 0.	1
Number of failures	When 0, no errors occurred during the re-encryption operation. If larger than 0, then investigate the NameNode log, and re-encrypt.	0

Column Name	Description	Sample Data
Last file Checkpointed	Identifies the current position of the re-encryption process in the encryption zone--in other words, the file that was most recently re-encrypted.	0

10. (***Optional***) After the re-encryption completes, you can confirm that the EDEK and EZ Key Version Name values have changed:

```
$ hdfs crypto –getFileEncryptionInfo –path /ez/f
{cipherSuite: {name: AES/CTR/NoPadding, algorithmBlockSize: 16}. cryptoProtocolVersion:
CryptoProtocolVersion{description='Encrytpion zones', version=2, unknownValue=null},
edek: 373c0c2e919c27e58c1c343f54233cbd,
iv: d129c913c8a34cde6371ec95edfb7337, keyName: key1, ezKeyVersionName:
ZMHfRoGKeXXgf0QzCX8q16NczIw2sq0rWRTOHS3YjCz }
```

Managing Re-encryption Operations

This section includes information that can help you manage various facets of the EDEK re-encryption process.

Cancelling Re-encryption

Only users with the HDFS Superuser privilege can cancel the EDEK re-encryption after the operation has started.

To cancel a re-encryption:

```
$ hadoop crypto -reencryptZone cancel -path <zone>
```

Rolling Keys During a Re-encryption Operation

While it is *not* recommended, it is possible to roll the encryption zone key version on the KMS while a re-encryption of that encryption zone is already in progress in the NameNode. The re-encryption is guaranteed to complete with all DEKs re-encrypted, with a key version equal to or later than the encryption zone key version when the re-encryption command was submitted. This means that, if initially the key version is rolled from v0 to v1, then a re-encryption command was submitted. If later on the KMS the key version is rolled again to v2, then all EDEKs will be at least re-encrypted to v1. To ensure that all EDEKs are re-encrypted to v2, submit another re-encryption command for the encryption zone.

Rolling keys during re-encryption is not recommended because of the potential negative impact on key management operations. Due to the asynchronous nature of re-encryption, there is no guarantee of when, exactly, the rolled encryption keys will take effect. Re-encryption can only guarantee that all EDEKs are re-encrypted at least on the EZ key version that existed when the re-encryption command is issued.

Throttling Re-encryption Operations

With the default operation settings, you will not typically need to throttle re-encryption operations. However, in cases of excessive performance impact due to the re-encryption of large numbers of files, advanced users have the option of throttling the operation so that the impact on the HDFS NameNode and KT KMS are minimized.

Specifically, you can throttle the operation to control the rate of the following:

- The number of EDEKs that the NameNode should send to the KMS to re-encrypt in a batch (dfs.namenode.reencrypt.batch.size)
- The number of threads in the NameNode that can run concurrently to contact the KMS. (dfs.namenode.reencrypt.edek.threads)
- Percentage of time the NameNode read-lock should be held by the re-encryption thread (dfs.namenode.reencrypt.throttle.limit.handler.ratio)

- Percentage of time the NameNode write-lock should be held by the re-encryption thread (dfs.namenode.reencrypt.throttle.limit.updater.ratio)

You can monitor the HDFS NameNode heap and CPU usage from Cloudera Manager.

Configuring the Key Management Server (KMS)

Hadoop Key Management Server (KMS) is a cryptographic key management server based on the Hadoop **KeyProvider** API. It provides a KeyProvider implementation client that interacts with the KMS using the HTTP REST API. Both the KMS and its client support HTTP SPNEGO Kerberos authentication and TLS/SSL-secured communication. The KMS is a Java-based web application that uses a preconfigured Tomcat server bundled with the Hadoop distribution.

For instructions on securing the KMS, see Securing the Key Management Server (KMS) on page 263.

Cloudera provides the following implementations of the Hadoop KMS:

- **Java KeyStore KMS** - The default Hadoop KMS included in CDH that uses a file-based Java KeyStore (JKS) for its backing keystore. For parcel-based installations, no additional action is required to install or upgrade the KMS. For package-based installations, you must install additional packages. For more information, see Installing and Upgrading Java KeyStore KMS. Cloudera strongly recommends not using Java Keystore KMS in production environments.
- **Key Trustee KMS** - A custom KMS that uses Cloudera Navigator Key Trustee Server for its backing keystore instead of the file-based Java KeyStore (JKS) used by the default Hadoop KMS. Cloudera strongly recommends using Key Trustee KMS in production environments to improve the security, durability, and scalability of your cryptographic key management. For more information about the architecture and components involved in encrypting data at rest for production environments, see Cloudera Navigator Data Encryption Overview and Data at Rest Encryption Reference Architecture on page 228. For instructions on installing and upgrading Key Trustee KMS, see:
  - Installing Key Trustee KMS
  - Upgrading Key Trustee KMS

  Also, integrating Key Trustee Server with Cloudera Navigator Key HSM provides an additional layer of protection.
- **Navigator KMS Services backed by Thales HSM** - A custom KMS that uses a supported Thales Hardware Security Module (HSM) as its backing keystore. This KMS service provides the highest level of key isolation to customers who require it.
- **Navigator KMS Services backed by Luna HSM** - A custom KMS that uses a supported Luna Hardware Security Module (HSM) as its backing keystore. This KMS provides the highest level of key isolation to customers who require it.

Configuring KMS High Availability

For Key Trustee KMS high availability, see Enabling Key Trustee KMS High Availability. Java KeyStore KMS does not support high availability.

Configuring the KMS Using Cloudera Manager

If you are using Cloudera Manager, you can view and edit the KMS configuration by navigating to the following pages, depending on the KMS implementation you are using:

- **Key Trustee KMS service** > **Configuration**
- **Java KeyStore KMS service** > **Configuration**

For more information on using Cloudera Manager to find and change configuration parameters, see Modifying Configuration Properties Using Cloudera Manager.

For instructions about configuring the KMS and its clients using the command line for package-based installations, continue reading:

Configuring the KMS Cache Using Cloudera Manager

By default, the KMS caches keys to reduce the number of interactions with the key provider. You can disable the cache by setting the `hadoop.kms.cache.enable` property to `false`.

The cache is only used with the `getCurrentKey()`, `getKeyVersion()` and `getMetadata()` methods.

For the `getCurrentKey()` method, entries are cached for a maximum of `30000` milliseconds to prevent stale keys.

For the `getKeyVersion()` method, entries are cached with a default inactivity timeout of `600000` milliseconds (10 minutes).

You can configure the cache and its timeout values by adding the following properties to **KMS service** > **Configuration** > **Advanced** > **Key Management Server Proxy Advanced Configuration Snippet (Safety Valve) for kms-site.xml**:

```
<property>
    <name>hadoop.kms.cache.enable</name>
    <value>true</value>
</property>

<property>
    <name>hadoop.kms.cache.timeout.ms</name>
    <value>600000</value>
</property>

<property>
    <name>hadoop.kms.current.key.cache.timeout.ms</name>
    <value>30000</value>
</property>
```

See Custom Configuration for more information on adding custom properties using the **Advanced Configuration Snippet (Safety Valve)** feature.

Configuring the Audit Log Aggregation Interval Using the Command Line

Audit logs are generated for `GET_KEY_VERSION`, `GET_CURRENT_KEY`, `DECRYPT_EEK`, and `GENERATE_EEK` operations.

Entries are aggregated by user, key, and operation for a configurable interval, after which the number of aggregated operations by the user for a given key is written to the audit log.

The interval is configured in milliseconds by adding the `hadoop.kms.aggregation.delay.ms` property to **KMS service** > **Configuration** > **Advanced** > **Key Management Server Proxy Advanced Configuration Snippet (Safety Valve) for kms-site.xml**:

```
<property>
    <name>hadoop.kms.aggregation.delay.ms</name>
    <value>10000</value>
</property>
```

For more information about adding custom properties using the **Advanced Configuration Snippet (Safety Valve)** feature, see Custom Configuration.

Configuring the Java KeyStore KMS Using the Command Line

> **Note:** Because Key Trustee KMS is supported only in Cloudera Manager deployments, the following command line instructions apply only to Java KeyStore KMS. For instructions about configuring Key Trustee KMS, see Configuring the KMS Using Cloudera Manager on page 259.

For instructions about configuring the Java KeyStore KMS and its clients using the command line for package-based installations, continue reading:

Configuring the Java KeyStore KMS KeyProvider Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

Configure the KMS backing KeyProvider properties in the `/etc/hadoop-kms/conf/kms-site.xml` configuration file:

```
<property>
    <name>hadoop.kms.key.provider.uri</name>
    <value>jceks://file@/${user.home}/kms.keystore</value>
</property>

<property>
    <name>hadoop.security.keystore.java-keystore-provider.password-file</name>
    <value>keystore_password_file</value>
</property>
```

If you do not specify the absolute path to the password file, you must include it in the Hadoop CLASSPATH.

Restart the KMS for configuration changes to take effect. See Starting and Stopping the Java KeyStore KMS Using the Command Line on page 262 for instructions.

Configuring the Java KeyStore KMS Cache Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

By default, the KMS caches keys to reduce the number of interactions with the key provider. You can disable the cache by setting the `hadoop.kms.cache.enable` property to `false`.

The cache is only used with the `getCurrentKey()`, `getKeyVersion()` and `getMetadata()` methods.

For the `getCurrentKey()` method, entries are cached for a maximum of `30000` milliseconds to prevent stale keys.

For the `getKeyVersion()` method, entries are cached with a default inactivity timeout of `600000` milliseconds (10 minutes).

The cache and its timeout values are configured using the following properties in the `/etc/hadoop-kms/conf/kms-site.xml` configuration file:

```
<property>
    <name>hadoop.kms.cache.enable</name>
    <value>true</value>
</property>

<property>
    <name>hadoop.kms.cache.timeout.ms</name>
    <value>600000</value>
</property>

<property>
    <name>hadoop.kms.current.key.cache.timeout.ms</name>
    <value>30000</value>
</property>
```

Configuring KMS Clients Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

To configure KMS clients, set the `hadoop.security.key.provider.path` property in `core-site.xml` or `hdfs-site.xml`. Specify the value in the format `kms://`*<scheme>*`@`*<kms_hosts>*`:`*<port>*`/kms`. Replace *<scheme>* with `http` or `https`, depending on whether you have configured TLS. Replace *<kms_hosts>* with a semicolon-separated list of the KMS hosts. Replace *<port>* with the port number on which the KMS is running (`16000` by default).

For example, for a KMS running on `http://localhost:16000/kms`, the KeyProvider URI is `kms://http@localhost:16000/kms`. For high availability KMS (Key Trustee KMS only) running on `https://kms01.example.com:16000/kms` and `https://kms02.example.com:16000/kms`, the KeyProvider URI is `kms://https@kms01.example.com;kms02.example.com:16000/kms`.

See the following for an excerpt from `core-site.xml`:

```
<property>
  <name>hadoop.security.key.provider.path</name>
  <value>kms://https@kms01.example.com;kms02.example.com:16000/kms</value>
</property>
```

Starting and Stopping the Java KeyStore KMS Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

To start or stop KMS use the `kms.sh` script. For example, to start the KMS:

```
$ sudo /usr/lib/hadoop-kms/sbin/kms.sh start
```

Running the script without parameters lists all possible parameters.

To use an `init` script to manage the KMS service, use your package manager to install the `hadoop-kms-server` package from the CDH repository. For example, for RHEL 6:

```
$ sudo yum install hadoop-kms-server
```

After installation, use the `service hadoop-kms-server` command to manage the KMS service.

Configuring the Audit Log Aggregation Interval Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

Audit logs are generated for `GET_KEY_VERSION, GET_CURRENT_KEY, DECRYPT_EEK`, and `GENERATE_EEK` operations.

Entries are aggregated by user, key, and operation for a configurable interval, after which the number of aggregated operations by the user for a given key is written to the audit log.

The interval is configured in milliseconds using the `hadoop.kms.aggregation.delay.ms` property:

```
<property>
    <name>hadoop.kms.aggregation.delay.ms</name>
    <value>10000</value>
</property>
```

Configuring the Embedded Tomcat Server Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

You can configure the embedded Tomcat server by using the `/etc/hadoop-kms/tomcat-conf/conf/server.xml.conf` file.

The following environment variables can be set in KMS `/etc/hadoop-kms/conf/kms-env.sh` script and can be used to alter the default ports and log directory:

- `KMS_HTTP_PORT`
- `KMS_ADMIN_PORT`
- `KMS_LOG`

Restart the KMS for the configuration changes to take effect.

Securing the Key Management Server (KMS)

Cloudera provides the following implementations of the Hadoop KMS:

- **Java KeyStore KMS** - The default Hadoop KMS included in CDH that uses a file-based Java KeyStore (JKS) for its backing keystore. For parcel-based installations, no additional action is required to install or upgrade the KMS. For package-based installations, you must install additional packages. For more information, see Installing and Upgrading Java KeyStore KMS. Cloudera strongly recommends not using Java Keystore KMS in production environments.
- **Key Trustee KMS** - A custom KMS that uses Cloudera Navigator Key Trustee Server for its backing keystore instead of the file-based Java KeyStore (JKS) used by the default Hadoop KMS. Cloudera strongly recommends using Key Trustee KMS in production environments to improve the security, durability, and scalability of your cryptographic key management. For more information about the architecture and components involved in encrypting data at rest for production environments, see Cloudera Navigator Data Encryption Overview and Data at Rest Encryption Reference Architecture on page 228. For instructions on installing and upgrading Key Trustee KMS, see:
  - Installing Key Trustee KMS
  - Upgrading Key Trustee KMS

  Also, integrating Key Trustee Server with Cloudera Navigator Key HSM provides an additional layer of protection.
- **Navigator KMS Services backed by Thales HSM** - A custom KMS that uses a supported Thales Hardware Security Module (HSM) as its backing keystore. This KMS service provides the highest level of key isolation to customers who require it.
- **Navigator KMS Services backed by Luna HSM** - A custom KMS that uses a supported Luna Hardware Security Module (HSM) as its backing keystore. This KMS provides the highest level of key isolation to customers who require it.

This topic contains information on securing the KMS using Kerberos, TLS/SSL communication, and access control lists (ACLs) for operations on encryption keys. Cloudera Manager instructions can be performed for both Key Trustee KMS and Java KeyStore KMS deployments. Command-line instructions apply only to Java KeyStore KMS deployments. Key Trustee KMS is not supported outside of Cloudera Manager. For more information, see Installing Key Trustee KMS.

Enabling Kerberos Authentication for the KMS

Enabling Kerberos Authentication for the KMS Using Cloudera Manager

Minimum Required Role: Full Administrator

To enable Kerberos for the KMS using Cloudera Manager:

1. Open the Cloudera Manager Admin Console and go to the KMS service.
2. Click **Configuration**.
3. Set the **Authentication Type** property to `kerberos`.
4. Click **Save Changes**.
5. Because Cloudera Manager does not automatically create the principal and keytab file for the KMS, you must run the **Generate Credentials** command manually. On the top navigation bar, go to **Administration** > **Security** > **Kerberos Credentials**and click **Generate Missing Credentials**.

> **Note:** This does not create a new Kerberos principal if an existing HTTP principal exists for the KMS host.

6. Return to the Home page by clicking the Cloudera Manager logo.
7. Click the ⏻ icon that is next to any stale services to invoke the cluster restart wizard.
8. Click **Restart Stale Services**.
9. Click **Restart Now**.
10. Click **Finish**.

Enabling Kerberos Authentication for the Java KeyStore KMS Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

Configure `/etc/krb5.conf` with information for your KDC server. Create an `HTTP` principal and keytab file for the KMS.

Configure `/etc/hadoop-kms/conf/kms-site.xml` with the following properties:

```
<property>
    <name>hadoop.kms.authentication.type</name>
    <value>kerberos</value>
</property>

<property>
    <name>hadoop.kms.authentication.kerberos.keytab</name>
    <value>${user.home}/kms.keytab</value>
</property>

<property>
    <name>hadoop.kms.authentication.kerberos.principal</name>
    <value>HTTP/localhost</value>
</property>

<property>
    <name>hadoop.kms.authentication.kerberos.name.rules</name>
    <value>DEFAULT</value>
</property>
```

Restart the KMS service for the configuration changes to take effect.

Configuring the Java KeyStore KMS Proxyuser Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

Each proxyuser must be configured in `/etc/hadoop-kms/conf/kms-site.xml` using the following properties:

```
<property>
    <name>hadoop.kms.proxyuser.#USER#.users</name>
    <value>*</value>
</property>

<property>
    <name>hadoop.kms.proxyuser.#USER#.groups</name>
    <value>*</value>
</property>

<property>
    <name>hadoop.kms.proxyuser.#USER#.hosts</name>
    <value>*</value>
</property>
```

where #USER# is the username of the proxyuser to be configured.

The `hadoop.kms.proxyuser.#USER#.users` property indicates the users that can be impersonated. The `hadoop.kms.proxyuser.#USER#.groups` property indicates the groups to which the users being impersonated must belong. At least one of these properties must be defined. If both are defined, the configured proxyuser can impersonate any user in the `users` list and any user belonging to a group listed in the `groups` list.

The `hadoop.kms.proxyuser.#USER#.hosts` property indicates the host from which the proxyuser can make impersonation requests. "*" means there are no restrictions for the #USER# regarding users, groups, or hosts.

Configuring TLS/SSL for the KMS

Configuring TLS/SSL for the KMS Using Cloudera Manager

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

The steps for configuring and enabling Hadoop TLS/SSL for the KMS are as follows:

1. Go to the KMS service.
2. Click **Configuration**.
3. In the Search field, type **TLS/SSL** to show the KMS TLS/SSL properties (in the **Key Management Server Default Group** > **Security** category).
4. Edit the following TLS/SSL properties according to your cluster configuration.

Table 21: KMS TLS/SSL Properties

Property	Description
Enable TLS/SSL for Key Management Server	Encrypt communication between clients and Key Management Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (TLS/SSL)).
Key Management Server TLS/SSL Server JKS Keystore File Location	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Key Management Server is acting as a TLS/SSL server. The keystore must be in JKS format.

Property	Description
Key Management Server TLS/SSL Server JKS Keystore File Password	The password for the Key Management Server JKS keystore file.
Key Management Server Proxy TLS/SSL Certificate Trust Store File	The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that Key Management Server Proxy might connect to. This is used when Key Management Server Proxy is the client in a TLS/SSL connection. This truststore must contain the certificates used to sign the services connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
Key Management Server Proxy TLS/SSL Certificate Trust Store Password	The password for the Key Management Server Proxy TLS/SSL Certificate Trust Store File. This password is not required to access the truststore; this field can be left blank. This password provides optional integrity checking of the file. The contents of truststores are certificates, and certificates are public information.

5. Click **Save Changes**.
6. Return to the Home page by clicking the Cloudera Manager logo.
7. Click the icon that is next to any stale services to invoke the cluster restart wizard.
8. Click **Restart Stale Services**.
9. Click **Restart Now**.
10. Click **Finish**.

Configuring TLS/SSL for the Java KeyStore KMS Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

To configure KMS to work over HTTPS, set the following properties in the `/etc/hadoop-kms/conf/kms_env.sh` script:

- `KMS_SSL_KEYSTORE_FILE`
- `KMS_SSL_KEYSTORE_PASS`
- `KMS_SSL_TRUSTSTORE_FILE`
- `KMS_SSL_TRUSTSTORE_PASS`

In the `/etc/hadoop-kms/tomcat-conf/conf/` directory, replace the `server.xml` file with the provided `ssl-server.xml` file.

Create a TLS/SSL certificate for the KMS. As the `kms` user, use the Java `keytool` command to create the TLS/SSL certificate:

```
$ keytool -genkey -alias tomcat -keyalg RSA
```

You are asked a series of questions in an interactive prompt. It creates the keystore file, which is named **.keystore** and located in the `kms` user home directory. The password you enter for the keystore must match the value of the KMS_SSL_KEYSTORE_PASS environment variable set in the `kms-env.sh` script in the configuration directory.

The answer to "What is your first and last name?" (CN) must be the hostname of the machine where the KMS will be running.

> **Note:** Restart the KMS for the configuration changes to take effect.

Configuring KMS Access Control Lists

Hadoop KMS supports a range of ACLs that control access to keys and key operations on a granular basis. ACLs can be used, for instance, to only grant users access to certain keys. Restricting HDFS superusers from access to key material is an important design requirement. This prevents a malicious superuser from having access to all the key material and all the encrypted data, and thus being able to decrypt everything.

There are two categories of KMS ACLs:

1. **KMS-wide:** These ACLs specify the types of operations a user can perform. They are configured using the `hadoop.kms.acl.<OPERATION>` and `hadoop.kms.blacklist.<OPERATION>` parameters. The operations are as follows:
   - `CREATE`
   - `DELETE`
   - `ROLLOVER`
   - `GET`
   - `GET_KEYS`
   - `GET_METADATA`
   - `SET_KEY_MATERIAL`
   - `GENERATE_EEK`
   - `DECRYPT_EEK`
2. **Key-specific:** These ACLs are set in a per-key basis. They are configured using the `default.key.acl.<OPERATION>`, `whitelist.key.acl.<OPERATION>`, and `key.acl.<key_name>.<OPERATION>` parameters. The operations and their programmatic equivalents are as follows:
   - `READ` - `getKeyVersion, getKeyVersions, getMetadata, getKeysMetadata, getCurrentKey`
   - `MANAGEMENT` - `createKey, deleteKey, rolloverNewVersion`
   - `GENERATE_EEK` - `generateEncryptedKey, warmUpEncryptedKeys`
   - `DECRYPT_EEK` - `decryptEncryptedKey`
   - `ALL` - All of the above

   The `default.key.acl.<OPERATION>` ACL applies to all keys for which an ACL has not been explicitly configured.

   If no ACL is configured for a specific key, *and* no default ACL is configured for the requested operation, access is denied.

   > **Note:** The default ACL does not support the `ALL` operation qualifier.

The KMS supports both whitelist and blacklist ACLs. Blacklist entries override whitelist entries. A user or group accessing the KMS is first checked for inclusion in the ACL for the requested operation and then checked for exclusion in the blacklist for the operation before access is granted.

The group membership used by ACL entries relies on the configured group mapping mechanism for HDFS. By default, group membership is determined on the local Linux system running the KMS service. If you have configured HDFS to use LDAP for group mapping, the group membership for the ACL entries is determined using the configured LDAP settings. For more information about LDAP-based group membership, see Configuring LDAP Group Mappings on page 343.

The ACL syntax for both blacklist and whitelist entries is as follows:

- Users only:

```
user1,user2,userN
```

> **Note:** There are no spaces following the commas separating the users in the list.

- Groups only:

```
nobody group1,group2,groupN
```

> **Note:** There is a space between nobody and the comma-separated group list. The nobody user, if it exists, must not have privileges to log in to or interact with the system. If you are uncertain about its access privileges, specify a different nonexistent user in its place.

- Users and Groups:

```
user1,user2,userN group1,group2,groupN
```

> **Note:** The comma-separated user list is separated from the comma-separated group list by a space.

Configuring KMS Access Control Lists Using Cloudera Manager

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

> **Important:** See related Known Issue and listed workaround: KMS and Key Trustee ACLs do not work in Cloudera Manager 5.3.

The KMS installation wizard includes an option to generate the recommended ACLs. To view or edit the ACLs:

1. Go to the KMS service.
2. Click **Configuration**.
3. In the Search field, type **acl** to show the **Key Management Server Advanced Configuration Snippet (Safety Valve) for kms-acls.xml** (in the **Key Management Server Default Group** category).
4. Add or edit the ACL properties according to your cluster configuration. See Recommended KMS Access Control List on page 269 for example ACL entries.
5. Click **Save Changes**.
6. Return to the Home page by clicking the Cloudera Manager logo.
7. Click the ⏻ icon that is next to any stale services to invoke the cluster restart wizard.
8. Click **Restart Stale Services**.
9. Click **Restart Now**.
10. Click **Finish**.

Configuring Java KeyStore KMS Access Control Lists Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

KMS ACLs are defined in the `/etc/hadoop-kms/conf/kms-acls.xml` configuration file. This file is hot-reloaded when it changes. See Recommended KMS Access Control List on page 269 for recommended ACL entries.

Recommended KMS Access Control List

Cloudera recommends the following ACL definition for secure production settings. Replace `keyadmin` and `keyadmingroup` with the user and group responsible for maintaining encryption keys.

> **Note:** If you are entering the ACL using Cloudera Manager, omit the surrounding `<configuration>` and `</configuration>` tags; Cloudera Manager adds this automatically.

```
<configuration>

<!--
  KMS ACLs control which users can perform various actions on the KMS,
  and which users and groups have access to which keys.

  This file has the following sections:
  * ACLs for KMS operations
  ** Access to specific KMS operations
  ** Blacklists for those specific operations
  * ACLs for keys
  ** Default ACLs for keys
  ** Whitelist ACLs for keys
  ** Key-specific ACLs
-->

<!--
  KMS ACLs that govern access to specific key operations. If access is not
  granted for an operation here, then the operation is forbidden, even if
  a key ACL allows it.

  The ACL value should be either a username or a username and group name
  separated by whitespace.

  A value of "*" (for the username or groupname) indicates that
  all users are granted access to that operation. Any operation for which
  there is no ACL or an empty (zero-length) ACL is treated as having an
  ACL with a value of "*". To disallow all users, add an ACL with a
  value of " ", a single space.

  Note: This convention applies only to the KMS-level ACLs beginning with
  'hadoop.kms.acl'.
-->

<property>
  <name>hadoop.kms.acl.CREATE</name>
  <value>keyadmin keyadmingroup</value>
  <description>
    ACL for create-key operations.
    If the user is not in the GET ACL, the key material is not returned
    as part of the response.
  </description>
</property>

<property>
  <name>hadoop.kms.acl.DELETE</name>
  <value>keyadmin keyadmingroup</value>
  <description>
    ACL for delete-key operations.
  </description>
</property>

<property>
  <name>hadoop.kms.acl.ROLLOVER</name>
  <value>keyadmin keyadmingroup</value>
  <description>
    ACL for rollover-key operations.
    If the user does is not in the GET ACL, the key material is not returned
    as part of the response.
```

```
  </description>
</property>

<property>
  <name>hadoop.kms.acl.GET</name>
  <value></value>
  <description>
    ACL for get-key-version and get-current-key operations.
  </description>
</property>

<property>
  <name>hadoop.kms.acl.GET_KEYS</name>
  <value>keyadmin keyadmingroup</value>
  <description>
    ACL for get-keys operations.
  </description>
</property>

<property>
  <name>hadoop.kms.acl.SET_KEY_MATERIAL</name>
  <value></value>
  <description>
    Complementary ACL for CREATE and ROLLOVER operations to allow the client
    to provide the key material when creating or rolling a key.
  </description>
</property>

<property>
  <name>hadoop.kms.acl.GENERATE_EEK</name>
  <value>hdfs supergroup</value>
  <description>
    ACL for generateEncryptedKey CryptoExtension operations.
  </description>
</property>

<!--
  KMS blacklists to prevent access to operations. These settings override the
  permissions granted by the KMS ACLs above.

  The blacklist value should be either a username or a username and group name
  separated by whitespace.

  A blank value indicates that no user is blacklisted from the operation. A
  value of "*" (for either the username or groupname) indicates that all users
  are blacklisted from the operation. Any operation for which there is no
  blacklist will be treated as having a blacklist with an empty value.
-->

<!--
  In this template the hdfs user is blacklisted for everything except
  GET_METADATA, GET_KEYS, and GENERATE_EEK. The GET and SET_KEY_MATERIAL
  operations are blacklisted for all users since Hadoop users should not
  need to perform those operations, and access to the key material should
  be as restricted as possible.
-->

<property>
  <name>hadoop.kms.blacklist.CREATE</name>
  <value>hdfs supergroup</value>
</property>

<property>
  <name>hadoop.kms.blacklist.DELETE</name>
  <value>hdfs supergroup</value>
</property>

<property>
  <name>hadoop.kms.blacklist.ROLLOVER</name>
  <value>hdfs supergroup</value>
</property>

<property>
```

```
    <name>hadoop.kms.blacklist.GET</name>
    <value>*</value>
  </property>

  <property>
    <name>hadoop.kms.blacklist.GET_KEYS</name>
    <value></value>
  </property>

  <property>
    <name>hadoop.kms.blacklist.SET_KEY_MATERIAL</name>
    <value>*</value>
  </property>

  <property>
    <name>hadoop.kms.blacklist.DECRYPT_EEK</name>
    <value>hdfs supergroup</value>
  </property>

  <property>
    <name>keytrustee.kms.acl.UNDELETE</name>
    <value></value>
    <description>
      ACL that grants access to the UNDELETE operation on all keys.
      Only used by Key Trustee KMS.
    </description>
  </property>

  <property>
    <name>keytrustee.kms.acl.PURGE</name>
    <value></value>
    <description>
      ACL that grants access to the PURGE operation on all keys.
      Only used by Key Trustee KMS.
    </description>
  </property>

  <!--
    Default key ACLs that govern access to key operations for key-operation pairs
    that do not have a specific key ACL already. Specific key ACLs will override
    the default key ACLs

    The ACL value should be either a username or a username and group name
    separated by whitespace.

    An empty value for an ACL indicates that no user is granted access to that
    operation. A value of "*" (for the username or groupname) indicates that
    all users are granted access to that operation. Any operation for which
    there is no ACL will be treated as having an ACL with an empty value.
  -->

  <property>
    <name>default.key.acl.MANAGEMENT</name>
    <value></value>
    <description>
      Default ACL that grants access to the MANAGEMENT operation on all keys.
    </description>
  </property>

  <property>
    <name>default.key.acl.GENERATE_EEK</name>
    <value></value>
    <description>
      Default ACL that grants access to the GENERATE_EEK operation on all keys.
    </description>
  </property>

  <property>
    <name>default.key.acl.DECRYPT_EEK</name>
    <value></value>
    <description>
      Default ACL that grants access to the DECRYPT_EEK operation on all keys.
    </description>
```

```
</property>

<property>
  <name>default.key.acl.READ</name>
  <value></value>
  <description>
    Default ACL that grants access to the READ operation on all keys.
  </description>
</property>

<!--
  Whitelist key ACLs that grant access to specific key operations. Any
  permissions granted here will be added to whatever permissions are granted
  by the specific key ACL or the default key ACL. Note that these whitelist
  ACLs grant access to operations on specific keys. If the operations
  themselves are not allowed because of the KMS ACLs/blacklists, then the
  operation will not be permitted, regardless of the whitelist settings.

  The ACL value should be either a username or a username and group name
  separated by whitespace.

  An empty value for an ACL indicates that no user is granted access to that
  operation. A value of "*" (for the username or groupname) indicates that
  all users are granted access to that operation. Any operation for which
  there is no ACL will be treated as having an ACL with an empty value.
-->

<property>
  <name>whitelist.key.acl.MANAGEMENT</name>
  <value>keyadmin keyadmingroup</value>
  <description>
    Whitelist ACL for MANAGEMENT operations for all keys.
  </description>
</property>

<property>
  <name>whitelist.key.acl.READ</name>
  <value>hdfs supergroup</value>
  <description>
    Whitelist ACL for READ operations for all keys.
  </description>
</property>

<property>
  <name>whitelist.key.acl.GENERATE_EEK</name>
  <value>hdfs supergroup</value>
  <description>
    Whitelist ACL for GENERATE_EEK operations for all keys.
  </description>
</property>

<property>
  <name>whitelist.key.acl.DECRYPT_EEK</name>
  <value>keyadmin keyadmingroup</value>
  <description>
    Whitelist ACL for DECRYPT_EEK operations for all keys.
  </description>
</property>

<!--
  Key ACLs that grant access to specific key operations. Any permissions
  granted here are added to whatever permissions are granted by the whitelists.
  The key ACL name should be key.acl.<keyname>.<OPERATION>.

  The ACL value should be either a username or a username and group name
  separated by whitespace.

  An empty value for an ACL indicates that no user is granted access to that
  operation. A value of "*" (for the username or groupname) indicates that
  all users are granted access to that operation. Any key operation for which
  there is no ACL will default to the default ACL for the operation.

  Normally adding users or groups for a specific key and DECRYPT_EEK is
```

```
    sufficient to allow access to data protected with HDFS data at rest
    encryption.
  -->

  <!--
    The following ACLs are required for proper functioning of services.
    CM does not create keys or encryption zones, however our best practices
    recommend encryption zones on certain directories. Below we assume that
    the user has followed our recommended naming scheme and named the keys
    according to our best practices: "hive-key" for the hive service,
    "hbase-key" for the hbase service, etc. If the key names are different,
    none of this will work out of the box, and you will need to edit these
    ACLs to match your key names.
  -->

  <property>
    <name>key.acl.hive-key.DECRYPT_EEK</name>
    <value>hive hive</value>
    <description>
      Gives the hive user and the hive group access to the key named "hive-key".
      This allows the hive service to read and write files in /user/hive/.
      Also note that the impala user ought to be a member of the hive group
      in order to enjoy this same access.
    </description>
  </property>

  <property>
    <name>key.acl.hive-key.READ</name>
    <value>hive hive</value>
    <description>
      Required because hive compares key strengths when joining tables.
    </description>
  </property>

  <property>
    <name>key.acl.hbase-key.DECRYPT_EEK</name>
    <value>hbase hbase</value>
    <description>
      Gives the hbase user and hbase group access to the key named "hbase-key".
      This allows the hbase service to read and write files in /hbase.
    </description>
  </property>

  <property>
    <name>key.acl.solr-key.DECRYPT_EEK</name>
    <value>solr solr</value>
    <description>
      Gives the solr user and solr group access to the key named "solr-key".
      This allows the solr service to read and write files in /solr.
    </description>
  </property>

  <property>
    <name>key.acl.mapred-key.DECRYPT_EEK</name>
    <value>mapred,yarn hadoop</value>
    <description>
      Gives the mapred user and mapred group access to the key named "mapred-key".
      This allows mapreduce to read and write files in /user/history.
      This is required by YARN.
    </description>
  </property>

  <property>
    <name>key.acl.hue-key.DECRYPT_EEK</name>
    <value>oozie,hue oozie,hue</value>
    <description>
      Gives the appropriate users and groups access to the key named "hue-key".
      This allows hue and oozie to read and write files in /user/hue.
      Oozie is required here because it will attempt to access workflows in
      /user/hue/oozie/workspaces.
    </description>
  </property>
```

```
<!-- This example is required if there are encryption zones on user’s home
     directories. -->
<!--
<property>
  <name>key.acl.username-key.DECRYPT_EEK</name>
  <value>username username,hive,hbase,solr,oozie,hue,yarn</value>
  <description>
    Designed to be placed on a key that protects the EZ /user/username,
    and assumes that the key name is also "username-key", this shows that
    a number of services may want to reach in to access data. Remove
    those are not needed for your use-case.
  </description>
</property>
-->

</configuration>
```

Configuring Java KeyStore KMS Delegation Tokens Using the Command Line

Important:

- Follow these command-line instructions on systems that do not use Cloudera Manager.
- This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

Configure KMS delegation token secret manager using the following properties:

```
  <property>
    <name>hadoop.kms.authentication.delegation-token.update-interval.sec</name>
    <value>86400</value>
    <description>
      How often the master key is rotated, in seconds. Default value 1 day.
    </description>
  </property>

  <property>
    <name>hadoop.kms.authentication.delegation-token.max-lifetime.sec</name>
    <value>604800</value>
    <description>
      Maximum lifetime of a delegation token, in seconds. Default value 7 days.
    </description>
  </property>

  <property>
    <name>hadoop.kms.authentication.delegation-token.renew-interval.sec</name>
    <value>86400</value>
    <description>
      Renewal interval of a delegation token, in seconds. Default value 1 day.
    </description>
  </property>

  <property>
    <name>hadoop.kms.authentication.delegation-token.removal-scan-interval.sec</name>
    <value>3600</value>
    <description>
      Scan interval to remove expired delegation tokens.
    </description>
  </property>
```

Migrating Keys from a Java KeyStore to Cloudera Navigator Key Trustee Server

You can migrate keys from an existing Java KeyStore (JKS) to Key Trustee Server to improve security, durability, and scalability. If you are using the Java KeyStore KMS service, and want to use Key Trustee Server as the backing key store for HDFS Transparent Encryption on page 235, use the following procedure.

This procedure assumes that the Java KeyStore (JKS) is on the same host as the new Key Trustee KMS service.

1. Stop the Java KeyStore KMS service.
2. Add and configure the Key Trustee KMS service, and configure HDFS to use it for its **KMS Service** setting. For more information about how to install Key Trustee KMS, see Installing Key Trustee KMS.
3. Restart the HDFS service and redeploy client configuration for this to take effect:
    a. **Home** > **Cluster-wide** > **Deploy Client Configuration**
4. Add the following to the **Key Management Server Proxy Advanced Configuration Snippet (Safety Valve) for kms-site.xml** (**Key Trustee KMS Service** > **Configuration** > **Category** > **Advanced**):

```
<property>
  <name>hadoop.kms.key.provider.uri</name>

<value>keytrustee://file@/var/lib/kms-keytrustee/keytrustee/.keytrustee/,jceks://file@/path/to/kms.keystore</value>

  <description>URI of the backing KeyProvider for the KMS</description>
</property>

<property>
  <name>hadoop.security.keystore.java-keystore-provider.password-file</name>
  <value>/tmp/password.txt</value>
  <description>Java KeyStore password file</description>
</property>
```

   If the Java KeyStore is *not* password protected, omit the `hadoop.security.keystore.java-keystore-provider.password-file` property.

5. Click **Save Changes** and restart the Key Trustee KMS service. If the Java KeyStore is *not* password protected, skip to step 7.
6. Create the file `/var/lib/keytrustee-kms/tomcat-deployment/webapps/kms/WEB-INF/classes/tmp/password.txt` and add the Java KeyStore password to it.
7. Change the ownership of `/var/lib/keytrustee-kms/tomcat-deployment/webapps/kms/WEB-INF/classes/tmp/password.txt` to `kms:kms`:

```
$ sudo chown kms:kms
/var/lib/keytrustee-kms/tomcat-deployment/webapps/kms/WEB-INF/classes/tmp/password.txt
```

8. From the host running the Key Trustee KMS service, if you have not configured Kerberos and TLS/SSL, run the following command:

```
$ curl -L -d "trusteeOp=migrate"
"http://kms01.example.com:16000/kms/v1/trustee/key/migrate?user.name=username&trusteeOp=migrate"
```

   If you have configured Kerberos and TLS/SSL, use the following command instead:

```
$ curl --negotiate -u : -L -d "trusteeOp=migrate"
"https://kms01.example.com:16000/kms/v1/trustee/key/migrate?user.name=username&trusteeOp=migrate"
 --cacert /path/to/kms/cert
```

9. Monitor `/var/log/kms-keytrustee/kms.log` and `/var/log/kms-keytrustee/kms-catalina.<date>.log` to verify that the migration is successful. You can also run `sudo -u <key_admin> hadoop key list` to verify that the keys are listed.

10. After you have verified that the migration is successful, remove the safety valve entry used in step 3 and restart the Key Trustee KMS service.

Configuring CDH Services for HDFS Encryption

This page contains recommendations for setting up HDFS Transparent Encryption on page 235 with various CDH services.

> **Important:** HDFS encryption does not support file transfer (reading, writing files) between zones through WebHDFS. For web-based file transfer between encryption zones managed by HDFS, use HttpFS with a load balancer instead.

> **Important:** Encrypting `/tmp` using HDFS encryption is *not* supported.

HBase

Recommendations

Make `/hbase` an encryption zone. Do not create encryption zones as subdirectories under `/hbase`, because HBase may need to rename files across those subdirectories. When you create the encryption zone, name the key `hbase-key` to take advantage of auto-generated KMS ACLs.

Steps

On a cluster without HBase currently installed, create the `/hbase` directory and make that an encryption zone.

On a cluster with HBase already installed, perform the following steps:

1. Stop the HBase service.
2. Move data from the `/hbase` directory to `/hbase-tmp`.
3. Create an empty `/hbase` directory and make it an encryption zone.
4. Distcp all data from `/hbase-tmp` to `/hbase`, preserving user-group permissions and extended attributes.
5. Start the HBase service and verify that it is working as expected.
6. Remove the `/hbase-tmp` directory.

KMS ACL Configuration for HBase

In the KMS ACLs, grant the `hbase` user and group `DECRYPT_EEK` permission for the HBase key:

```
<property>
  <name>key.acl.hbase-key.DECRYPT_EEK</name>
  <value>hbase hbase</value>
  </description>
</property>
```

Hive

HDFS encryption has been designed so that files cannot be moved from one encryption zone to another or from encryption zones to unencrypted directories. Therefore, the landing zone for data when using the `LOAD DATA INPATH` command must always be inside the destination encryption zone.

To use HDFS encryption with Hive, ensure you are using *one* of the following configurations:

Single Encryption Zone

With this configuration, you can use HDFS encryption by having all Hive data inside the same encryption zone. In Cloudera Manager, configure the Hive Scratch Directory (`hive.exec.scratchdir`) to be inside the encryption zone.

Recommended HDFS Path: `/user/hive`

To use the auto-generated KMS ACLs, make sure you name the encryption key `hive-key`.

For example, to configure a single encryption zone for the entire Hive warehouse, you can rename `/user/hive` to `/user/hive-old`, create an encryption zone at `/user/hive`, and then `distcp` all the data from `/user/hive-old` to `/user/hive`.

In Cloudera Manager, configure the Hive Scratch Directory (`hive.exec.scratchdir`) to be inside the encryption zone by setting it to `/user/hive/tmp`, ensuring that permissions are `1777` on `/user/hive/tmp`.

Multiple Encryption Zones

With this configuration, you can use encrypted databases or tables with different encryption keys. To read data from read-only encrypted tables, users must have access to a temporary directory that is encrypted at least as strongly as the table.

For example:

1. Configure two encrypted tables, `ezTbl1` and `ezTbl2`.
2. Create two new encryption zones, `/data/ezTbl1` and `/data/ezTbl2`.
3. Load data to the tables in Hive using `LOAD` statements.

For more information, see Changed Behavior after HDFS Encryption is Enabled on page 277.

Other Encrypted Directories

- **`LOCALSCRATCHDIR`**: The MapJoin optimization in Hive writes HDFS tables to a local directory and then uploads them to the distributed cache. To ensure these files are encrypted, either disable MapJoin by setting `hive.auto.convert.join` to `false`, or encrypt the *local* Hive Scratch directory (`hive.exec.local.scratchdir`) using Cloudera Navigator Encrypt.
- **`DOWNLOADED_RESOURCES_DIR`**: JARs that are added to a user session and stored in HDFS are downloaded to `hive.downloaded.resources.dir` on the HiveServer2 local filesystem. To encrypt these JAR files, configure Cloudera Navigator Encrypt to encrypt the directory specified by `hive.downloaded.resources.dir`.
- **NodeManager Local Directory List:** Hive stores JARs and MapJoin files in the distributed cache. To use MapJoin or encrypt JARs and other resource files, the `yarn.nodemanager.local-dirs` YARN configuration property must be configured to a set of encrypted local directories on all nodes.

Changed Behavior after HDFS Encryption is Enabled

- Loading data from one encryption zone to another results in a copy of the data. Distcp is used to speed up the process if the size of the files being copied is higher than the value specified by `HIVE_EXEC_COPYFILE_MAXSIZE`. The minimum size limit for `HIVE_EXEC_COPYFILE_MAXSIZE` is 32 MB, which you can modify by changing the value for the `hive.exec.copyfile.maxsize` configuration property.
- When loading data to encrypted tables, Cloudera strongly recommends using a landing zone inside the same encryption zone as the table.
  - **Example 1: Loading unencrypted data to an encrypted table** - Use one of the following methods:
    - If you are loading new unencrypted data to an encrypted table, use the `LOAD DATA ...` statement. Because the source data is not inside the encryption zone, the `LOAD` statement results in a copy. For this reason, Cloudera recommends landing data that you need to encrypt inside the destination encryption zone. You can use `distcp` to speed up the copying process if your data is inside HDFS.

   – If the data to be loaded is already inside a Hive table, you can create a new table with a LOCATION inside an encryption zone as follows:

```
CREATE TABLE encrypted_table [STORED AS] LOCATION ... AS SELECT * FROM <unencrypted_table>
```

   The location specified in the CREATE TABLE statement must be inside an encryption zone. Creating a table pointing LOCATION to an unencrypted directory does not encrypt your source data. You must copy your data to an encryption zone, and then point LOCATION to that zone.

 – **Example 2: Loading encrypted data to an encrypted table** - If the data is already encrypted, use the CREATE TABLE statement pointing LOCATION to the encrypted source directory containing the data. This is the fastest way to create encrypted tables.

```
CREATE TABLE encrypted_table [STORED AS] LOCATION ... AS SELECT * FROM
<encrypted_source_directory>
```

- Users reading data from encrypted tables that are read-only must have access to a temporary directory which is encrypted with at least as strong encryption as the table.
- Temporary data is now written to a directory named .hive-staging in each table or partition
- Previously, an INSERT OVERWRITE on a partitioned table inherited permissions for new data from the existing partition directory. With encryption enabled, permissions are inherited from the table.

KMS ACL Configuration for Hive

When Hive joins tables, it compares the encryption key strength for each table. For this operation to succeed, you must configure the KMS ACLs to allow the hive user and group READ access to the Hive key:

```
<property>
  <name>key.acl.hive-key.READ</name>
  <value>hive hive</value>
</property>
```

If you have restricted access to the GET_METADATA operation, you must grant permission for it to the hive user or group:

```
<property>
  <name>hadoop.kms.acl.GET_METADATA</name>
  <value>hive hive</value>
</property>
```

If you have disabled HiveServer2 Impersonation on page 131 (for example, to use Apache Sentry), you must configure the KMS ACLs to grant DECRYPT_EEK permissions to the hive user, as well as any user accessing data in the Hive warehouse.

Cloudera recommends creating a group containing all Hive users, and granting DECRYPT_EEK access to that group.

For example, suppose user jdoe (home directory /user/jdoe) is a Hive user and a member of the group hive-users. The encryption zone (EZ) key for /user/jdoe is named jdoe-key, and the EZ key for /user/hive is hive-key. The following ACL example demonstrates the required permissions:

```
<property>
  <name>key.acl.hive-key.DECRYPT_EEK</name>
  <value>hive hive-users</value>
</property>

<property>
  <name>key.acl.jdoe-key.DECRYPT_EEK</name>
  <value>jdoe,hive</value>
</property>
```

If you have enabled HiveServer2 impersonation, data is accessed by the user submitting the query or job, and the user account (jdoe in this example) may still need to access data in their home directory. In this scenario, the required permissions are as follows:

```
<property>
  <name>key.acl.hive-key.DECRYPT_EEK</name>
  <value>nobody hive-users</value>
</property>

<property>
  <name>key.acl.jdoe-key.DECRYPT_EEK</name>
  <value>jdoe</value>
</property>
```

Hue

Recommendations

Make /user/hue an encryption zone because Oozie workflows and other Hue-specific data are stored there by default. When you create the encryption zone, name the key hue-key to take advantage of auto-generated KMS ACLs.

Steps

On a cluster without Hue currently installed, create the /user/hue directory and make it an encryption zone.

On a cluster with Hue already installed:

1. Create an empty /user/hue-tmp directory.
2. Make /user/hue-tmp an encryption zone.
3. DistCp all data from /user/hue into /user/hue-tmp.
4. Remove /user/hue and rename /user/hue-tmp to /user/hue.

KMS ACL Configuration for Hue

In the KMS ACLs, grant the hue and oozie users and groups DECRYPT_EEK permission for the Hue key:

```
<property>
  <name>key.acl.hue-key.DECRYPT_EEK</name>
  <value>oozie,hue oozie,hue</value>
</property>
```

Impala

Recommendations

- If HDFS encryption is enabled, configure Impala to encrypt data spilled to local disk.
- In releases lower than Impala 2.2.0 / CDH 5.4.0, Impala does not support the LOAD DATA statement when the source and destination are in different encryption zones. If you are running an affected release and need to use LOAD DATA with HDFS encryption enabled, copy the data to the table's encryption zone prior to running the statement.
- Use Cloudera Navigator to lock down the local directory where Impala UDFs are copied during execution. By default, Impala copies UDFs into /tmp, and you can configure this location through the --local_library_dir startup flag for the impalad daemon.
- Limit the rename operations for internal tables once encryption zones are set up. Impala cannot do an ALTER TABLE RENAME operation to move an internal table from one database to another, if the root directories for those databases are in different encryption zones. If the encryption zone covers a table directory but not the parent directory associated with the database, Impala cannot do an ALTER TABLE RENAME operation to rename an internal table, even within the same database.

- Avoid structuring partitioned tables where different partitions reside in different encryption zones, or where any partitions reside in an encryption zone that is different from the root directory for the table. Impala cannot do an `INSERT` operation into any partition that is not in the same encryption zone as the root directory of the overall table.
- If the data files for a table or partition are in a different encryption zone than the HDFS trashcan, use the `PURGE` keyword at the end of the `DROP TABLE` or `ALTER TABLE DROP PARTITION` statement to delete the HDFS data files immediately. Otherwise, the data files are left behind if they cannot be moved to the trashcan because of differing encryption zones. This syntax is available in Impala 2.3 / CDH 5.5 and higher.

Steps

Start every `impalad` process with the `--disk_spill_encryption=true` flag set. This encrypts all spilled data using AES-256-CFB. Set this flag by selecting the **Disk Spill Encryption** checkbox in the Impala configuration (**Impala service** > **Configuration** > **Category** > **Security**).

> **Important:** Impala does not selectively encrypt data based on whether the source data is already encrypted in HDFS. This results in at most 15 percent performance degradation when data is spilled.

KMS ACL Configuration for Impala

Cloudera recommends making the `impala` user a member of the `hive` group, and following the ACL recommendations in KMS ACL Configuration for Hive on page 278.

MapReduce and YARN

MapReduce v1

Recommendations

MRv1 stores both history and logs on local disks by default. Even if you do configure history to be stored on HDFS, the files are not renamed. Hence, no special configuration is required.

MapReduce v2 (YARN)

Recommendations

Make `/user/history` a single encryption zone, because history files are moved between the `intermediate` and done directories, and HDFS encryption does not allow moving encrypted files across encryption zones. When you create the encryption zone, name the key `mapred-key` to take advantage of auto-generated KMS ACLs.

Steps

On a cluster with MRv2 (YARN) installed, create the `/user/history` directory and make that an encryption zone.

If `/user/history` already exists and is not empty:

1. Create an empty `/user/history-tmp` directory.
2. Make `/user/history-tmp` an encryption zone.
3. DistCp all data from `/user/history` into `/user/history-tmp`.
4. Remove `/user/history` and rename `/user/history-tmp` to `/user/history`.

KMS ACL Configuration for MapReduce

In the KMS ACLs, grant DECRYPT_EEK permission for the MapReduce key to the `mapred` and `yarn` users and the hadoop group:

```
<property>
  <name>key.acl.mapred-key.DECRYPT_EEK</name>
  <value>mapred,yarn hadoop</value>
  </description>
</property>
```

Search

Recommendations

Make `/solr` an encryption zone. When you create the encryption zone, name the key `solr-key` to take advantage of auto-generated KMS ACLs.

Steps

On a cluster without Solr currently installed, create the `/solr` directory and make that an encryption zone.

On a cluster with Solr already installed:

1. Create an empty `/solr-tmp` directory.
2. Make `/solr-tmp` an encryption zone.
3. DistCp all data from `/solr` into `/solr-tmp`.
4. Remove `/solr`, and rename `/solr-tmp` to `/solr`.

KMS ACL Configuration for Search

In the KMS ACLs, grant the `solr` user and group DECRYPT_EEK permission for the Solr key:

```
<property>
  <name>key.acl.solr-key.DECRYPT_EEK</name>
  <value>solr solr</value>
  </description>
</property>
```

Spark

Recommendations

- By default, application event logs are stored at `/user/spark/applicationHistory`, which can be made into an encryption zone.
- Spark also optionally caches its JAR file at `/user/spark/share/lib` (by default), but encrypting this directory is not required.
- Spark does not encrypt shuffle data. To do so, configure the Spark local directory, `spark.local.dir` (in Standalone mode), to reside on an encrypted disk. For YARN mode, make the corresponding YARN configuration changes.

KMS ACL Configuration for Spark

In the KMS ACLs, grant DECRYPT_EEK permission for the Spark key to the `spark` user and any groups that can submit Spark jobs:

```
<property>
  <name>key.acl.spark-key.DECRYPT_EEK</name>
  <value>spark spark-users</value>
</property>
```

Sqoop

Recommendations

- **For Hive support:** Ensure that you are using Sqoop with the `--target-dir` parameter set to a directory that is inside the Hive encryption zone. For more details, see Hive on page 276.
- **For append/incremental support:** Make sure that the `sqoop.test.import.rootDir` property points to the same encryption zone as the `--target-dir` argument.
- **For HCatalog support:** No special configuration is required.

Cloudera Navigator Key Trustee Server

Cloudera Navigator Key Trustee Server is an enterprise-grade crytographic key storage and management system used by Cloudera Navigator Encrypt on page 309 that separates encryption keys from the data, thus ensuring data is protected even if unauthorized users gain access to the storage media. It enables your cluster to meet the strictest data security regulations. Furthermore, Navigator Key Trustee Server can be integrated with a hardware security module (HSM) to provide the highest level of security for your keys. See Cloudera Navigator Key HSM on page 301 for details.

In addition, Navigator Key Trustee Server can be used by other cluster components. For example, HDFS Transparent Encryption on page 235 can use Navigator Key Trustee Server (KTS) as its backing key store (for Hadoop KMS, instead of the default Java KeyStore) for better security and scalability. See Migrating Keys from a Java KeyStore to Cloudera Navigator Key Trustee Server on page 275 for more information about using Navigator KTS with HDFS encryption.

Important: Cloudera recommends that each cluster use its own KTS instance. Although sharing a single KTS across clusters is technically possible, it is neither approved nor supported for security reasons—specifically, the increased security risks associated with single point of failure for encryption keys used by multiple clusters.

After Installing Cloudera Navigator Key Trustee Server, follow the steps below to manage the system:

Backing Up and Restoring Key Trustee Server and Clients

Key Trustee Server high availability applies to read operations only. If either Key Trustee Server fails, the client automatically retries fetching keys from the functioning server. New write operations (for example, creating new encryption keys) are not allowed unless both Key Trustee Servers are operational.

If a Key Trustee Server fails catastrophically, you must restore it from backup to a new host with the same hostname and IP address as the failed host. Cloudera does not support PostgreSQL promotion to convert a passive Key Trustee Server to an active Key Trustee Server.

Cloudera strongly recommends regularly backing up Key Trustee Server databases and configuration files. Because these backups contain encryption keys and encrypted deposits, you must ensure that your backup repository is as secure as the Key Trustee Server.

You must also back up client configuration files and keys for Key Trustee Server clients, such as Key Trusteee KMS and Navigator Encrypt clients.

Note: In an HA configuration, the backup need only be performed on one of the hosts for Key Trustee Server and the Key Trustee KMS.

Backing Up Key Trustee Server and Key Trustee KMS Using the ktbackup.sh Script

Key Trustee Server releases 5.7 and higher include a script, `ktbackup.sh`, to simplify and automate backing up Key Trustee Server. Key Trustee KMS releases 5.7 and higher include the same script for backing up Key Trustee KMS.

When run on a Key Trustee Server host, the script creates a tarball containing the Key Trustee Server private GPG keys and the PostgreSQL database. When run on a Key Trustee KMS host, the script creates a tarball containing the Key Trustee KMS private GPG keys and configuration file.

To preserve the security of the backup, you must specify a GPG recipient. Because this recipient is the only entity that can decrypt the backup, the recipient must be someone authorized to access the Key Trustee Server database, such as a key administrator.

Creating and Importing a GPG Key for Encrypting and Decrypting Backups

If the key administrator responsible for backing up and restoring Key Trustee Server and Key Trustee KMS does not already have a GPG key pair, they can create one using the `gpg --gen-key` command. The following example demonstrates this procedure:

> **Note:** By default, `gpg --gen-key` fails at the password prompt if you have logged in to your user account with the su command. You must log in to the SSH session with the user account for which you want to generate the GPG key pair.

```
[john.doe@backup-host ~]$ gpg --gen-key
gpg (GnuPG) 2.0.14; Copyright (C) 2009 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: John Doe
Email address: john.doe@example.com
Comment: Key Trustee Backup
You selected this USER-ID:
    "John Doe (Key Trustee Backup) <john.doe@example.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.

can't connect to `/home/john.doe/.gnupg/S.gpg-agent': No such file or directory
gpg-agent[10638]: directory `/home/john.doe/.gnupg/private-keys-v1.d' created
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /home/john.doe/.gnupg/trustdb.gpg: trustdb created
gpg: key 0936CB67 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048R/0936CB67 2016-02-10
      Key fingerprint = CE57 FDED 3AFE E67D 2041  9EBF E64B 7D00 0936 CB67
uid                  John Doe (Key Trustee Backup) <john.doe@example.com>
sub   2048R/52A6FC5C 2016-02-10
```

After the GPG key pair is generated, you can export the public key:

```
[john.doe@backup-host ~]$ gpg --export 'John Doe' --armor > /path/to/johndoe.pub
```

Copy the public key (`johndoe.pub` in this example) to the Key Trustee Server or Key Trustee KMS host, and import it into the service account keyring (`keytrustee` for Key Trustee Server and `kms` for Key Trustee KMS):

- On the Key Trustee Server host:

```
$ sudo -u keytrustee gpg --import /path/to/johndoe.pub
```

- On the Key Trustee KMS host:

```
$ sudo -u kms gpg --import /path/to/johndoe.pub
```

Running the ktbackup.sh Script

You must run `ktbackup.sh` as the service account. The location of the script depends on the service and installation method. See the following table for the script location and default service account for package- and parcel-based installations for Key Trustee Server and Key Trustee KMS.

Table 22: Backup Script Locations

Service	Service Account	Parcel-Based Installation	Package-Based Installation
Key Trustee Server	`keytrustee`	`/opt/cloudera/parcels/KEYTRUSTEE_SERVER/bin/ktbackup.sh`	`/usr/bin/ktbackup.sh`
Key Trustee KMS	`kms`	`/opt/cloudera/parcels/KEYTRUSTEE/bin/ktbackup.sh`	`/usr/share/keytrustee-keyprovider/bin/ktbackup.sh`

The following table lists the command options for `ktbackup.sh`.

Table 23: Command Options for ktbackup.sh

Command Option	Description
`-c, --confdir=CONFDIR`	Specifies the Key Trustee configuration directory. Defaults to `/var/lib/keytrustee/.keytrustee` for parcel-based Key Trustee Server. For Key Trustee KMS and package-based Key Trustee Server, you must specify this option.
`--database-port=PORT`	Specifies the Key Trustee Server database port. Defaults to `11381` for parcel-based installations. For package-based Key Trustee Server installations, you must specify this option.
`--gpg-recipient=GPG_RECIPIENT`	Specifies the GPG recipient. The backup is encrypted with the public key of the specified recipient. The GPG recipient public key must be imported into the service account keyring before running the script. See Creating and Importing a GPG Key for Encrypting and Decrypting Backups on page 284 for more information.
`--cleartext`	Outputs an unencrypted tarball. To preserve the security of the cryptographic keys, *do not* use this option in production environments.
`--output=DIR`	Specifies the output directory for the tarball. Defaults to `/var/lib/keytrustee` for parcel-based Key Trustee

Command Option	Description
	Server. For Key Trustee KMS and package-based Key Trustee Server, you must specify this option.
`--roll=n`	Deletes backups older than the last *n* backups from the directory specified by the `--output` paramter. For example, if you have 10 backups, specifying `--roll=10` creates a new backup (11 backups total) and then delete the oldest backup. Specifying `--roll=1` creates a new backup and then deletes all other backups.
`-q, --quiet`	Suppresses console log messages and, if successful, returns only the backup tarball file path. This is useful for automating backups.
`--verbose`	Outputs additional log messages to the console for debugging.

The following examples demonstrate the command usage for different scenarios:

- To back up a parcel-based Key Trustee Server, specifying the GPG recipient by name:

```
$ sudo -u keytrustee /opt/cloudera/parcels/KEYTRUSTEE_SERVER/bin/ktbackup.sh
--gpg-recipient='John Doe'
```

- To back up a package-based Key Trustee KMS, specifying the GPG recipient by email:

```
$ sudo -u kms /usr/share/keytrustee-keyprovider/bin/ktbackup.sh -c
/var/lib/kms-keytrustee/keytrustee/.keytrustee --output=/var/lib/kms-keytrustee
--gpg-recipient=john.doe@example.com
```

- To back up a package-based Key Trustee Server with the database running on a nondefault port (12345 in this example):

```
$ sudo -u keytrustee ktbackup.sh --database-port=12345
--gpg-recipient=john.doe@example.com
```

Automating Backups Using cron

You can schedule automatic backups of Key Trustee Server or Key Trustee KMS using the `cron` scheduling utility.

Create a `crontab` entry using the following commands:

- For Key Trustee Server:

  1. Edit the `crontab` by running the following command:

```
$ sudo -u keytrustee crontab -e
```

  2. Add the following entry to run the backup script every 30 minutes. This example is for a parcel-based installation of Key Trustee Server. See the Backup Script Locations table for the package-based script location.

```
*/30 * * * * /opt/cloudera/parcels/KEYTRUSTEE_SERVER/bin/ktbackup.sh --gpg-recipient='John
 Doe' --quiet --output=/tmp/backups --roll=10
```

  Run `man 5 crontab` to see the `crontab` man page for details on using `cron` to schedule backups at different intervals.

- For Key Trustee KMS:

1. Edit the `crontab` by running the following command:

```
$ sudo -u kms crontab -e
```

2. Add the following entry to run the backup script every 30 minutes. This example is for a parcel-based installation of Key Trustee KMS. See the Backup Script Locations table for the package-based script location.

```
*/30 * * * * /opt/cloudera/parcels/KEYTRUSTEE/bin/ktbackup.sh --gpg-recipient='John Doe'
 --quiet --output=/tmp/backups --roll=10
```

Run `man 5 crontab` to see the `crontab` man page for details on using `cron` to schedule backups at different intervals.

Backing Up Key Trustee Server Manually

Use this procedure for both parcel-based and package-based installations.

If you have deployed Cloudera Navigator Key Trustee Server High Availability, perform these steps on both the active and passive Key Trustee Servers. The following procedure references the default database port and location; if you modified these settings during installation, replace the database and port with your values.

1. Back up the Key Trustee Server database:

   - For Key Trustee Server 3.8:

```
$ su - postgres
$ pg_dump -c -p 5432 keytrustee | zip --encrypt keytrustee-db.zip -
```

   - For Key Trustee Server 5.4 and higher:

```
$ su - keytrustee
$ pg_dump -c -p 11381 keytrustee | zip --encrypt keytrustee-db.zip -
```

   The `--encrypt` option prompts you to create a password used to encrypt the zip file. This password is required to decrypt the file.

   For parcel-based installations, you must set environment variables after switching to the `keytrustee` user:

```
$ su - keytrustee
$ export PATH=$PATH:/opt/cloudera/parcels/KEYTRUSTEE_SERVER/PG_DB/opt/postgres/9.3/bin
$ export
LD_LIBRARY_PATH=/opt/cloudera/parcels/KEYTRUSTEE_SERVER/PG_DB/opt/postgres/9.3/lib
$ pg_dump -c -p 11381 keytrustee | zip --encrypt keytrustee-db.zip -
```

2. Back up the Key Trustee Server configuration directory (`/var/lib/keytrustee/.keytrustee`):

```
$ zip -r --encrypt keytrustee-conf.zip /var/lib/keytrustee/.keytrustee
```

   The `--encrypt` option prompts you to create a password used to encrypt the zip file. This password is required to decrypt the file.

3. Move the backup files (*keytrustee-db.zip* and *keytrustee-conf.zip*) to a secure location.

Backing Up Key Trustee Server Clients

Cryptographic keys stored in Key Trustee Server are encrypted by clients before they are sent to Key Trustee Server. The primary clients for Key Trustee Server are Key Trustee KMS and Navigator Encrypt. Cloudera strongly recommends backing up regularly the configuration files and GPG keys for Key Trustee Server clients. See Backing Up Key Trustee Server and Key Trustee KMS Using the ktbackup.sh Script on page 283 for instructions on backing up Key Trustee KMS using the provided backup script.

> **Warning:** Failure to back up these files can result in irretrievable data loss. For example, encryption zone keys used for HDFS Transparent Encryption on page 235 are encrypted by the KMS before being stored in Key Trustee Server. A catastrophic failure of the KMS with no backup causes all HDFS data stored in encryption zones to become permanently irretrievable.

To prevent permanent data loss, regularly back up the following directories on each client that stores objects in Key Trustee Server:

Table 24: Key Trustee Server Client Configuration Directories

Key Trustee Server Client	Directories to Back Up
Key Trustee KMS	`/var/lib/kms-keytrustee`
Navigator Encrypt	`/etc/navencrypt`

Restoring Key Trustee Server

When restoring the Key Trustee Server database from backup, keep in mind that any keys or deposits created after the backup are not restored. If you are using Key Trustee Server high availability, you can restore the Active Key Trustee Server from the Passive Key Trustee Server. This restores all keys that were successfully written to the Passive Key Trustee Server before the failure.

The procedure to restore Key Trustee Server is different for parcel-based than for package-based installations. For more information about parcels, see Parcels.

Restoring Key Trustee Server in Parcel-Based Installations

> **Note:** These instructions apply to Key Trustee Servers deployed using parcels. For package-based deployments, skip to the Restoring Key Trustee Server in Package-Based Installations on page 289 section.

If you have deployed Cloudera Navigator Key Trustee Server High Availability, perform these steps on both the active and passive Key Trustee Servers. The following procedures assume the default database port and location; if you modified these settings during installation, replace the database and port with your custom values.

If the Key Trustee Server host has failed completely, remove the host from the cluster and add a new host using Cloudera Manager:

1. Remove the failed host from the cluster. See Deleting Hosts for instructions.
2. Add a new host with the same hostname and IP address as the failed host to the cluster. See Adding a Host to the Cluster for instructions.

   > **Important:** Make sure that the replacement host uses the same operating system version as the failed host.

3. Install Key Trustee Server on the new host. See Installing Cloudera Navigator Key Trustee Server for instructions. Make sure to install the same Key Trustee Server version as the failed host.

After you have provisioned a new host and installed Key Trustee Server (or if you are restoring the database or configuration on the original host), restore the database and configuration directory. If your backups were created using the `ktbackup.sh` script, skip to Restoring Key Trustee Server and Key Trustee KMS from ktbackup.sh Backups on page 290. If you need to restore the Active Key Trustee Server from the Passive Key Trustee Server, skip to Restoring Active Key Trustee Server from Passive Key Trustee Server on page 292.

If your backups were created manually using the pg_dump command, do the following:

1. Copy or move the backup files (*keytrustee-db.zip* and *keytrustee-conf.zip*) to the Key Trustee Server host.

2. Start the PostgreSQL server:

```
$ sudo ktadmin db --start --pg-rootdir /var/lib/keytrustee/db --background
```

3. Restore the Key Trustee Server database:

```
$ su - keytrustee
$ export PATH=$PATH:/opt/cloudera/parcels/KEYTRUSTEE_SERVER/PG_DB/opt/postgres/9.3/bin
$ export
LD_LIBRARY_PATH=/opt/cloudera/parcels/KEYTRUSTEE_SERVER/PG_DB/opt/postgres/9.3/lib
$ unzip -p /path/to/keytrustee-db.zip | psql -p 11381 -d keytrustee
```

If the zip file is encrypted, you are prompted for the password to decrypt the file.

4. Restore the Key Trustee Server configuration directory:

```
$ su - keytrustee
$ cd /var/lib/keytrustee
$ unzip /path/to/keytrustee-conf.zip
```

If the zip file is encrypted, you are prompted for the password to decrypt the file.

5. Stop the PostgreSQL server:

```
$ sudo ktadmin db --stop --pg-rootdir /var/lib/keytrustee/db
```

6. Start the Key Trustee Server service in Cloudera Manager (**Key Trustee Server service** > **Actions** > **Start**).
7. Restart the Key Trustee KMS service in Cloudera Manager (**Key Trustee KMS service** > **Actions** > **Restart**).
8. Remove the backup files (*keytrustee-db.zip* and *keytrustee-conf.zip*) from the Key Trustee Server host.

Restoring Key Trustee Server in Package-Based Installations

If you have deployed Cloudera Navigator Key Trustee Server High Availability, perform these steps on both the active and passive Key Trustee Servers. The following procedures assume the default database port and location; if you modified these settings during installation, replace the database and port with your custom values.

If the Key Trustee Server host has failed completely, provision a new host with the same hostname and IP address as the failed host, and re-install Key Trustee Server. See Installing Cloudera Navigator Key Trustee Server for instructions.

> **Important:** Make sure to install the same operating system and Key Trustee Server versions as the failed host.

After you have provisioned a new host and installed Key Trustee Server (or if you are restoring the database or configuration on the original host), restore the database and configuration directory. If your backups were created using the `ktbackup.sh` script, skip to Restoring Key Trustee Server and Key Trustee KMS from ktbackup.sh Backups on page 290. If you need to restore the Active Key Trustee Server from the Passive Key Trustee Server, skip to Restoring Active Key Trustee Server from Passive Key Trustee Server on page 292.

If your backups were created manually using the pg_dump command, do the following:

1. Copy or move the backup files (*keytrustee-db.zip* and *keytrustee-conf.zip*) to the Key Trustee Server host.
2. Change the file ownership on the backup files to `keytrustee:keytrustee`:

```
$ sudo chown keytrustee:keytrustee /path/to/keytrustee*.zip
```

3. Restore the Key Trustee Server database:

```
$ su - keytrustee
$ unzip -p /path/to/keytrustee-db.zip | psql -p 11381 -d keytrustee
```

If the zip file is encrypted, you are prompted for the password to decrypt the file.

4. Restore the Key Trustee Server configuration directory:

```
$ cd /var/lib/keytrustee
$ unzip /path/to/keytrustee-conf.zip
```

If the zip file is encrypted, you are prompted for the password to decrypt the file.

5. Start the Key Trustee Server service:

   - RHEL 6-compatible: `$ sudo service keytrusteed start`
   - RHEL 7-compatible: `$ sudo systemctl start keytrusteed`

6. Remove the backup files (*keytrustee-db.zip* and *keytrustee-conf.zip*) from the Key Trustee Server host.

Restoring Key Trustee Server and Key Trustee KMS from ktbackup.sh Backups

After installing Key Trustee Server or Key Trustee KMS on a new host after a failure, or if you need to restore accidentally deleted keys on the same host, use the following procedure to restore Key Trustee Server or Key Trustee KMS from backups generated by the `ktbackup.sh` script.

1. Decrypt the backup tarball using the private key of the GPG recipient specified in the backup command by running the following command as the GPG recipient user account. The GPG recipient private key must be available on the Key Trustee Server or Key Trustee KMS host on which you run this command.

```
$ gpg -d -o /path/to/decrypted/backup.tar /path/to/encrypted/tarball
```

2. Verify the decrypted tarball using the `tar tvf /path/to/decrypted/backup.tar` command. For example:

```
$ tar tvf kts_bak_kts01_example_com_2016-02-10_11-14-37.tar
drwx------ keytrustee/keytrustee 0 2016-02-09 16:43 var/lib/keytrustee/.keytrustee/
-rw------- keytrustee/keytrustee 434 2016-02-09 16:43
var/lib/keytrustee/.keytrustee/keytrustee.conf
-rw------- keytrustee/keytrustee 1280 2016-02-09 16:43
var/lib/keytrustee/.keytrustee/trustdb.gpg
-rw------- keytrustee/keytrustee 4845 2016-02-09 16:43
var/lib/keytrustee/.keytrustee/secring.gpg
-rw------- keytrustee/keytrustee  600 2016-02-09 16:43
var/lib/keytrustee/.keytrustee/random_seed
drwx------ keytrustee/keytrustee    0 2016-02-09 16:40
var/lib/keytrustee/.keytrustee/.ssl/
-rw------- keytrustee/keytrustee 1708 2016-02-09 16:40
var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-pk.pem
-rw------- keytrustee/keytrustee 1277 2016-02-09 16:40
var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.pem
-rw------- keytrustee/keytrustee 2263 2016-02-09 16:43
var/lib/keytrustee/.keytrustee/pubring.gpg
-rw-r--r-- keytrustee/keytrustee  457 2016-02-09 16:43
var/lib/keytrustee/.keytrustee/logging.conf
-rw------- keytrustee/keytrustee 2263 2016-02-09 16:43
var/lib/keytrustee/.keytrustee/pubring.gpg~
-rw------- keytrustee/keytrustee  157 2016-02-09 16:40
var/lib/keytrustee/.keytrustee/gpg.conf
-rw-r--r-- keytrustee/keytrustee 47752 2016-02-10 11:14
var/lib/keytrustee/kts_bak_kts01_example_com_2016-02-10_11-14-37.sql
```

3. Restore the files to their original locations, using this command for both Key Trustee Server and Key Trustee KMS backups:

```
$ tar xvf /path/to/decrypted/backup.tar -C /
```

4. **(Key Trustee Server Only)** Drop and re-create the `keytrustee` PostgreSQL database, and restore the database from the backup.

- For parcel-based installations:

```
$ su - keytrustee
$ source /opt/cloudera/parcels/KEYTRUSTEE_SERVER/meta/keytrustee_env.sh
$ /opt/cloudera/parcels/KEYTRUSTEE_SERVER/PG_DB/opt/postgres/9.3/bin/psql -p 11381
psql (9.3.6)
Type "help" for help.

keytrustee=# \list
                                     List of databases
    Name    |   Owner    | Encoding |   Collate    |    Ctype     |     Access privileges
------------+------------+----------+--------------+--------------+--------------------------

 keytrustee | keytrustee | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres   | keytrustee | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0  | keytrustee | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/keytrustee
     +
            |            |          |             |             |
keytrustee=CTc/keytrustee
 template1  | keytrustee | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/keytrustee
     +
            |            |          |             |             |
keytrustee=CTc/keytrustee
(4 rows)

keytrustee=# \c postgres;
You are now connected to database "postgres" as user "keytrustee".
postgres=# drop database keytrustee;
DROP DATABASE
postgres=# create database keytrustee;
CREATE DATABASE
postgres=# \q
$ sudo -u keytrustee
/opt/cloudera/parcels/KEYTRUSTEE_SERVER/PG_DB/opt/postgres/9.3/bin/psql -p 11381 -f
/var/lib/keytrustee/kts_bak_kts01_example_com_2016-02-10_11-14-37.sql
```

- For package-based installations:

```
$ su - keytrustee
$ psql -p 11381
psql (9.3.6)
Type "help" for help.

keytrustee=# \list
                                     List of databases
    Name    |   Owner    | Encoding |   Collate    |    Ctype     |     Access privileges
------------+------------+----------+--------------+--------------+--------------------------

 keytrustee | keytrustee | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres   | keytrustee | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0  | keytrustee | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/keytrustee
     +
            |            |          |             |             |
keytrustee=CTc/keytrustee
 template1  | keytrustee | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/keytrustee
     +
            |            |          |             |             |
keytrustee=CTc/keytrustee
(4 rows)

keytrustee=# \c postgres;
You are now connected to database "postgres" as user "keytrustee".
postgres=# drop database keytrustee;
DROP DATABASE
postgres=# create database keytrustee;
CREATE DATABASE
postgres=# \q
$ sudo -u keytrustee psql -p 11381 -f
/var/lib/keytrustee/kts_bak_kts01_example_com_2016-02-10_11-14-37.sql
```

5. Restart Key Trustee Server.

- **Using Cloudera Manager: Key Trustee Server service** > **Actions** > **Restart**
- **Using the Command Line:** Run the following command on the Key Trustee Server hosts:

```
$ sudo service keytrusteed restart      #RHEL 6-compatible
$ sudo systemctl restart keytrusteed    #RHEL 7-compatible
```

6. Restart the Key Trustee KMS service in Cloudera Manager (**Key Trustee KMS service** > **Actions** > **Restart**).

Restoring Active Key Trustee Server from Passive Key Trustee Server

If the Active Key Trustee Server fails, and you do not have a backup, you can restore it from the Passive Key Trustee Server using the following procedure. You can also use this procedure if you need to restore keys that were successfully written to the Passive Key Trustee Server, but are not included in the most recent backup.

The following procedure assumes you have installed Key Trustee Server on the replacement host and (if you are using Cloudera Manager) added the Key Trustee Server service. For instructions on installing Key Trustee Server, see Installing Cloudera Navigator Key Trustee Server

1. Copy the Key Trustee Server database from the Passive Key Trustee Server host to the new Active Key Trustee Server host. Run the following command on the Passive Key Trustee Server host:

```
$ sudo rsync --exclude recovery.conf -a /var/lib/keytrustee/db
root@kts01.example.com:/var/lib/keytrustee/
```

   Replace *kts01.example.com* with the hostname of the new Active Key Trustee Server.

2. Make sure that the `recovery.conf` file did not get copied to the Active Key Trustee Server (for example, if there was a typo in your `rsync` command). Run the following command on the Active Key Trustee Server host:

```
$ sudo ls -l /var/lib/keytrustee/db/recovery.conf
```

   If the file exists on the Active Key Trustee Server host, delete it. Make sure you are on the Active Key Trustee Server host before deleting the file. Do not delete the `recovery.conf` file on the Passive Key Trustee Server host.

3. Copy the configuration directory from the Passive Key Trustee Server host to the new Active Key Trustee Server host. Run the following command on the Passive Key Trustee Server host:

```
$ sudo rsync --exclude .ssl --exclude '*.pid' -a /var/lib/keytrustee/.keytrustee
root@kts01.example.com:/var/lib/keytrustee/
```

   Replace *kts01.example.com* with the hostname of the new Active Key Trustee Server.

4. Create the `logs` directory and make sure it is owned by the `keytrustee` user and group:

```
$ sudo mkdir /var/lib/keytrustee/logs
$ sudo chown keytrustee:keytrustee /var/lib/keytrustee/logs
```

5. **(Cloudera Manager only)** Generate the Key Trustee Server keyring: **Key Trustee Server service** > **Actions** > **Generate Key Trustee Server Keyring**
6. Set up the database on the Active Key Trustee Server host.
   - **Using Cloudera Manager: Key Trustee Server service** > **Actions** > **Set Up Key Trustee Server Database**
   - **Using the Command Line:**

```
$ sudo ktadmin --confdir /var/lib/keytrustee db --port 11381 --pg-rootdir
/var/lib/keytrustee/db --bootstrap --slave kts02.example.com
```

   Replace *kts02.example.com* with the hostname of the Passive Key Trustee Server.

7. Start the database.

- **Using Cloudera Manager: Key Trustee Server service** > **Instances** > **Active Database** > **Actions** > **Start this Active Database**
- **Using the Command Line:** Run the following command on the Active Key Trustee Server host:

```
$ sudo ktadmin --confdir /var/lib/keytrustee db --port 11381 --pg-rootdir
/var/lib/keytrustee/db --bootstrap --slave kts02.example.com
```

Replace *kts02.example.com* with the hostname of the Passive Key Trustee Server.

8. Enable synchronous replication.

- **Using Cloudera Manager: Key Trustee Server service** > **Actions** > **Setup Enable Synchronous Replication in HA mode**
- **Using the Command Line:** Run the following command on the Active Key Trustee Server host:

```
$ sudo ktadmin --confdir /var/lib/keytrustee enable-synchronous-replication
```

Replace *kts02.example.com* with the hostname of the Passive Key Trustee Server.

9. Restart the Active Key Trustee Server.

- **Using Cloudera Manager: Key Trustee Server service** > **Actions** > **Restart**
- **Using the Command Line:** Run the following command on the Active Key Trustee Server host:

```
$ sudo service keytrusteed restart      #RHEL 6-compatible
$ sudo systemctl restart keytrusteed    #RHEL 7-compatible
```

10. Restart the Key Trustee KMS service in Cloudera Manager (**Key Trustee KMS service** > **Actions** > **Restart**).

Initializing Standalone Key Trustee Server

If you are configuring high availability Key Trustee Server, skip this step and proceed to Cloudera Navigator Key Trustee Server High Availability. Cloudera strongly recommends configuring high availability for Key Trustee Server.

Initializing Standalone Key Trustee Server Using Cloudera Manager

Important: If you are using SSH software other than OpenSSH, the initialization fails. To prevent this, pre-create the SSH key before continuing:

```
$ sudo -u keytrustee ssh-keygen -t rsa -f /var/lib/keytrustee/.ssh/id_rsa
```

For new installations, use the **Set up HDFS Data At Rest Encryption** wizard and follow the instructions in Enabling HDFS Encryption Using the Wizard on page 240. When prompted, deselect the **Enable High Availability** option to proceed in standalone mode.

To set up Key Trustee Server manually, add the Key Trustee Server service to your cluster, following the instructions in Adding a Service. When customizing role assignments, assign only the Active Key Trustee Server and Active Database roles.

Important: You *must* assign the Key Trustee Server and Database roles to the same host. Key Trustee Server does not support running the database on a different host.

For parcel-based Key Trustee Server releases 5.8 and higher, Cloudera Manager automatically backs up Key Trustee Server (using the `ktbackup.sh` script) after adding the Key Trustee Server service. It also schedules automatic backups using `cron`. For package-based installations, you must manually back up Key Trustee Server and configure a `cron` job.

Cloudera Manager configures `cron` to run the backup script hourly. The latest 10 backups are retained in `/var/lib/keytrustee` in cleartext. For information about using the backup script and configuring the `cron` job (including how to encrypt backups), see Backing Up Key Trustee Server and Key Trustee KMS Using the ktbackup.sh Script on page 283.

Initializing Standalone Key Trustee Server Using the Command Line

To initialize a standalone Key Trustee Server, run the following commands on the Key Trustee Server:

> **Important:** For Key Trustee Server 5.4.0 and higher, the `ktadmin init-master` command is deprecated. Use the `ktadmin init` command instead. If you are using SSH software other than OpenSSH, the initialization fails. To prevent this, pre-create the SSH key before continuing:
>
> ```
> $ sudo -u keytrustee ssh-keygen -t rsa /var/lib/keytrustee/.ssh/id_rsa
> ```

```
$ sudo ktadmin init --external-address keytrustee.example.com
$ sudo ktadmin db --bootstrap --port 11381 --pg-rootdir /var/lib/keytrustee/db
## For RHEL/CentOS 7, use 'sudo systemctl [stop|start] <service_name>' instead of 'sudo
 service <service_name> [stop|start]' ##
$ sudo service keytrustee-db stop
$ sudo service keytrustee-db start
$ sudo service keytrusteed start
$ sudo chkconfig keytrustee-db on
$ sudo chkconfig keytrusteed on
```

Replace *`keytrustee.example.com`* with the fully qualified domain name (FQDN) of the Key Trustee Server. Cloudera recommends using the default `/var/lib/keytrustee/db` directory for the PostgreSQL database.

To use a different port for the database, modify the `ktadmin init` and `ktadmin db` commands as follows:

```
$ sudo ktadmin init --external-address keytrustee.example.com --db-connect
postgresql://localhost:<port>/keytrustee?host=/tmp
$ sudo ktadmin db --bootstrap --port <port> --pg-rootdir /var/lib/keytrustee/db
```

If you specify a database directory other than `/var/lib/keytrustee/db`, create or edit the `/etc/sysconfig/keytrustee-db` file and add the following line:

```
ARGS="--pg-rootdir /path/to/db"
```

The `ktadmin init` command initializes the Key Trustee configuration directory (`/var/lib/keytrustee/.keytrustee` by default) and generates a self-signed certificate that Key Trustee Server uses for HTTPS communication.

The `ktadmin db --bootstrap` command initializes the database in the directory specified by the `--pg-rootdir` parameter.

The `sudo service keytrustee-db stop` and `sudo service keytrustee-db stop` commands restart the Key Trustee Server database.

The `sudo service keytrusteed start` command starts Key Trustee Server.

> **Note:** The `/etc/init.d/postgresql` script does not work when the PostgreSQL database is started by Key Trustee Server, and cannot be used to monitor the status of the database. Use `/etc/init.d/keytrustee-db` instead.

(Optional) Replace Self-Signed Certificate with CA-Signed Certificate

> **Important:** Key Trustee Server certificates must be issued to the fully qualified domain name (FQDN) of the Key Trustee Server host. If you are using CA-signed certificates, ensure that the generated certificates use the FQDN, and not the short name.

If you have a CA-signed certificate for Key Trustee Server, see Managing Key Trustee Server Certificates on page 298 for instructions on how to replace the self-signed certificate.

Configuring a Mail Transfer Agent for Key Trustee Server

The Key Trustee Server requires a mail transfer agent (MTA) to send email. Cloudera recommends Postfix, but you can use any MTA that meets your needs.

To configure Postfix for local delivery, run the following commands:

```
export
KEYTRUSTEE_SERVER_PK="/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-pk.pem"
export
KEYTRUSTEE_SERVER_CERT="/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.pem"
export
KEYTRUSTEE_SERVER_CA="/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-ca.pem"
export KEYTRUSTEE_SERVER_HOSTNAME="$(hostname -f)"  # or adjust as required
postconf -e 'mailbox_command ='
postconf -e 'smtpd_sasl_local_domain ='
postconf -e 'smtpd_sasl_auth_enable = yes'
postconf -e 'smtpd_sasl_security_options = noanonymous'
postconf -e 'broken_sasl_auth_clients = yes'
postconf -e 'smtpd_recipient_restrictions =
permit_sasl_authenticated,permit_mynetworks,reject_unauth_destination'
postconf -e 'inet_interfaces = localhost'
postconf -e 'smtp_tls_security_level = may'
postconf -e 'smtpd_tls_security_level = may'
```

Start the Postfix service and ensure that it starts at boot:

```
$ service postfix restart
$ sudo chkconfig --level 235 postfix on
```

For information on installing Postfix or configuring a relay host, see the Postfix documentation.

Verifying Cloudera Navigator Key Trustee Server Operations

Verify that the installation was successful by running the following command on all Key Trustee Servers. The output should be similar to the following. If high availability is enabled, the output should be identical on all Key Trustee Servers:

```
$ curl -k https://keytrustee.example.com:11371/?a=fingerprint
4096R/4EDC46882386C827E20DEEA2D850ACA33BEDB0D1
```

Replace `keytrustee.example.com` with the fully qualified domain name (FQDN) of each Key Trustee Server you are validating.

Managing Key Trustee Server Organizations

Organizations allow you to configure Key Trustee for use in a multi-tenant environment. Using the `keytrustee-orgtool` utility, you can create organizations and administrators for multiple organizations. Organization administrators can then approve or deny the registration of clients, depending on the registration method.

The `keytrustee-orgtool` Utility

`keytrustee-orgtool` is a command-line utility for administering organizations. The `keytrustee-orgtool` command must be run as the root user.

The following table explains the various `keytrustee-orgtool` commands and parameters. Run `keytrustee-orgtool --help` to view this information at the command line.

Table 25: Usage for `keytrustee-orgtool`

Operation	Usage	Description
Add	`keytrustee-orgtool add [-h] –n` *`name`* `–c` *`contacts`*	Adds a new organization and administrators for the organization.
List	`keytrustee-orgtool list`	Lists current organizations, including the authorization secret, all administrators, the organization creation date, and the organization expiration date.
Disable client	`keytrustee-orgtool disable-client [-h] -–fingerprint` *`fingerprint`*	Disables a client that has already been activated by the organization administrator.
Enable client	`keytrustee-orgtool enable-client [-h] -–fingerprint` *`fingerprint`*	Enables a client that has requested activation but has not yet been approved by the organization administrator.
Set authorization Code	`keytrustee-orgtool set-auth [-h] –n` *`name`* `–s` *`secret`*	Sets the authorization code to a new string, or to blank to allow automatic approvals without the code.

Create Organizations

Each new Key Trustee tenant needs its own organization. You can create new organizations using the `keytrustee-orgtool add` command. For example, to create a new organization for the Disaster Recovery group and add two administrators, Finn and Jake:

```
$ keytrustee-orgtool add -n disaster-recov -c finn@example.com,jake@example.com
```

When adding organizations:

- Do not use spaces or special characters in the organization name. Use hyphens or underscores instead.
- Do not use spaces between email addresses (when adding multiple administrators to an organization). Use a comma to separate email addresses, without any space (as shown in the example above).

Each contact email address added when creating the organization receives a notification email, as detailed below.

Once an organization exists, use the `keytrustee-orgtool add` command to add new administrators to the organization. For example, to add an administrator to the `disaster-recov` organization:

```
keytrustee-orgtool add -n disaster-recov -c marceline@example.com
```

Note: You cannot remove contacts from an organization with the `keytrustee-orgtool` utility.

List Organizations

After creating an organization, verify its existence with the `keytrustee-orgtool list` command. This command lists details for all existing organizations. The following is the entry for the `disaster-recov` organization created in the example:

```
"disaster-recov": {
           "auth_secret": "/qFiICsyYqMLhdTznNY3Nw==",
           "contacts": [
            "finn@example.com",
            "jake@example.com"
        ],
        "creation": "2013-12-02T09:55:21",
        "expiration": "9999-12-31T15:59:59",
        "key_info": null,
        "name": "disaster-recov",
        "state": 0,
        "uuid": "xY3Z8xCwMuKZMiTYJa0mZOdhMVdxhyCUOc6vSNc9I8X"
    }
```

Change the Authorization Code

When an organization is created, an authorization code is automatically generated. When you run the `keytrustee-orgtool list` command, the code is displayed in the `auth_secret` field. To register with a Key Trustee Server, the client must have the authorization code along with the organization name. To set a new `auth_secret`, run the following command:

```
$ keytrustee-orgtool set-auth -n disaster-recov -s ThisISAs3cr3t!
```

Run the `keytrustee-orgtool list` command again, and confirm the updated `auth_secret` field:

```
"disaster-recov": {
          "auth_secret": "ThisISAs3cr3t!",
          "contacts": [
            "finn@example.com",
            "jake@example.com"
        ],
        "creation": "2013-12-02T09:55:21",
        "expiration": "9999-12-31T15:59:59",
        "key_info": null,
        "name": "disaster-recov",
        "state": 0,
        "uuid": "xY3Z8xCwMuKZMiTYJa0mZOdhMVdxhyCUOc6vSNc9I8X"
    }
```

If you do not want to use an authorization code, set the `auth_secret` field to an empty string:

```
$ keytrustee-orgtool set-auth -n disaster-recov -s ""
```

Cloudera recommends requiring an authorization code.

Notification Email and GPG Keys

Whenever an administrator is added to an organization, the Key Trustee Server sends an automated email message (subject: “KeyTrustee Contact Registration”) to the newly added administrator:

```
Hello, this is an automated message from your Cloudera keytrustee Server.

Welcome to Cloudera keytrustee! You have been listed as an administrator contact
for keytrustee services at your organization [test-org]. As an administrator,
you may be contacted to authorize the activation of new keytrustee clients.

We recommend that you register a GPG public key for secure administration of
your clients. To do so, visit the link below and follow the instructions.
```

```
https://keytrustee01.example.com:11371/?q=CnRV6u0nbm7zB07BQEpXCXsN0QJFBz684uC0lcHMoWL

This link will expire in 12 hours, at Thu Sep  3 00:08:25 2015 UTC.
```

Cloudera recommends that an organization's administrators:

- Register the GPG public key by following the link contained in the notification email. Registering the GPG public key is optional, but if you choose to register your public key:
  - Complete the process within 12 hours, before the link expires.
  - Upload the entire key, including the BEGIN and END strings, as shown here:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.1 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQGiBDkHP3URBACkWGsYh43pkXU9wj/X1G67K8/DSrl85r7dNtHNfLL/ewil10k2
q8saWJn26QZPsDVqdUJMOdHfJ6kQTAt9NzQbgcVrxLYNfgeBsvkHF/POtnYcZRgL
tZ6syBBWs8JB4xt5V09iJSGAMPUQE8Jpdn2aRXPApdoDw179LM8Rq6r+gwCg5ZZa
. . .
-----END PGP PUBLIC KEY BLOCK-----
```

- Import the Key Trustee Server's public GPG key to verify that the server is the sender.

The organization's administrators are notified by email when new clients are registered to the Key Trustee Server using the mail transfer agent (as discussed in Configuring a Mail Transfer Agent for Key Trustee Server on page 295). However, if the server does not have access to email, you can use a local system mail address, such as `username@hostname`, where `hostname` is the system hostname and `username` is a valid system user. If you use a local system email address, be sure to regularly monitor the email box.

Managing Key Trustee Server Certificates

Transport Layer Security (TLS) certificates are used to secure communication with Key Trustee Server. By default, Key Trustee Server generates self-signed certificates when it is first initialized. Cloudera strongly recommends using certificates signed by a trusted Certificate Authority (CA).

Generating a New Certificate

1. Generate a new certificate signing request (CSR):

```
$ openssl req -new -key keytrustee_private_key.pem -out new.csr
```

   Replace *keytrustee_private_key.pem* with the filename of the private key. You can reuse the existing private key or generate a new private key in accordance with your company policies. For existing auto-generated self-signed certificates, the private key file is located at `/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-pk.pem`.

2. Generate a new certificate from the CSR:
   - For a CA-signed certificate, submit the CSR to the CA, and they will provide a signed certificate.
   - To generate a new self-signed certificate, run the following command:

```
$ openssl x509 -req -days 365 -in new.csr -signkey keytrustee_private_key.pem \
-out new_keytrustee_certificate.pem
```

Replacing Key Trustee Server Certificates

Use the following procedure if you need to replace an existing certificate for the Key Trustee Server. For example, you can use this procedure to replace the auto-generated self-signed certificate with a CA-signed certificate, or to replace an expired certificate.

> **Note:** Key Trustee Server supports password-protected private keys, but not password-protected certificates.

1. After Generating a New Certificate on page 298, back up the original certificate and key files:

```
$ sudo cp -r /var/lib/keytrustee/.keytrustee/.ssl /var/lib/keytrustee/.keytrustee/.ssl.bak
```

2. **(CA-Signed Certificates Only)** Provide the root or intermediate CA certificate:

```
$ sudo mv /path/to/rootca.pem
/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-ca.pem
```

3. Make sure that the certificate and key files are owned by the `keytrustee` user and group, with file permissions set to `600`:

```
$ sudo chown keytrustee:keytrustee /path/to/new/certificate.pem
/path/to/new/private_key.pem
$ sudo chmod 600 /path/to/new/certificate.pem /path/to/new/private_key.pem
```

4. Update the Key Trustee Server configuration with the location of the new certificate and key files:

   - Using Cloudera Manager:

     1. Go to the Key Trustee Server service.
     2. Click the **Configuration** tab.
     3. Select **Category** > **Security**.
     4. Edit the following properties to specify the location of the new certificate and key files. If the private keys are not password protected, leave the password fields empty.

        - **Active Key Trustee Server TLS/SSL Server Private Key File (PEM Format)**
        - **Active Key Trustee Server TLS/SSL Server Certificate File (PEM Format)**
        - **Active Key Trustee Server TLS/SSL Private Key Password**
        - **Passive Key Trustee Server TLS/SSL Server Private Key File (PEM Format)**
        - **Passive Key Trustee Server TLS/SSL Server Certificate File (PEM Format)**
        - **Passive Key Trustee Server TLS/SSL Private Key Password**

     5. Click **Save Changes** to commit the changes.

   - Using the command line:

     1. Edit `/var/lib/keytrustee/.keytrustee/keytrustee.conf` on the active and passive Key Trustee Server hosts and modify the SSL_CERTIFICATE and SSL_PRIVATE_KEY parameters as follows:

```
    "SSL_CERTIFICATE": "/path/to/new/certificate.pem",
    "SSL_PRIVATE_KEY": "/path/to/new/private_key.pem"
```

If the private key is password protected, add the following entry:

```
    "SSL_PRIVATE_KEY_PASSWORD_SCRIPT": "/path/to/password_script [arguments]"
```

Replace */path/to/password_script [arguments]* with the path to a script (and any necessary command arguments) that returns the password for the private key file. If you do not want to create a script, you can use a simple command, such as `echo -n password`. For example:

```
    "SSL_PRIVATE_KEY_PASSWORD_SCRIPT": "/bin/echo -n password"
```

Keep in mind that this method can expose the private key password in plain text to anyone who can view the `/var/lib/keytrustee/.keytrustee/keytrustee.conf` file.

5. Restart Key Trustee Server:
    - **Using Cloudera Manager:** Restart the Key Trustee Server service (**Key Trustee Server service** > **Actions** > **Restart**).
    - **Using the Command Line:** Restart the Key Trustee Server daemon:
        - RHEL 6-compatible: `$ sudo service keytrusteed restart`
        - RHEL 7-compatible: `$ sudo systemctl restart keytrusteed`
6. If you are using the Key Trustee KMS service in Cloudera Manager for HDFS Transparent Encryption on page 235, update the Java KeyStore (JKS) used on the Key Trustee KMS host:
    a. Download the new certificate to the Key Trustee KMS host:

```
$ echo -n | openssl s_client -connect keytrustee01.example.com:11371 \
| sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > /tmp/keytrustee_certificate.pem
```

    b. Delete the existing keystore entry for *keytrustee01.example.com*:

```
$ keytool -delete -alias key_trustee_alias_name -keystore /path/to/truststore -v
```

    c. Add the new keystore entry for *keytrustee01.example.com*:

```
$ keytool -import -trustcacerts -alias keytrustee01.example.com \
-file /tmp/keytrustee_certificate.pem -keystore /path/to/truststore
```

    d. Restart the Key Trustee KMS service in Cloudera Manager.
7. If you are using Key HSM, update the Key Trustee Server and Key HSM configuration:
    a. Run the `keyhsm trust` command, using the path to the new certificate:

```
$ sudo keyhsm trust /path/to/new/key_trustee_server/cert
```

    b. Run the `ktadmin keyhsm` command, using the `--client-certfile` and `--client-keyfile` options to specify the location of the new certificate file and private key:

```
$ sudo ktadmin keyhsm --server https://keyhsm01.example.com:9090 --client-certfile
/path/to/new/key_trustee_server/cert --client-keyfile
/path/to/new/key_trustee_server/private_key
```

Cloudera Navigator Key HSM

Cloudera Navigator Key HSM allows Cloudera Navigator Key Trustee Server to seamlessly integrate with a hardware security module (HSM). Key HSM enables Key Trustee Server to use an HSM as a root of trust for cryptographic keys, taking advantage of Key Trustee Server's policy-based key and security asset management capabilities while at the same time satisfying existing, internal security requirements regarding treatment of cryptographic materials.

For instructions on installing Key HSM, see Installing Cloudera Navigator Key HSM.

For instructions on configuring Key HSM, continue reading:

Initializing Navigator Key HSM

Before initializing Navigator Key HSM, verify that the HSM is properly configured and accessible from the Key HSM host, and that the HSM client libraries are installed on the Key HSM host:

- **SafeNet Luna**

  Install the SafeNet Luna client. No additional configuration is needed.

- **SafeNet KeySecure**

  Extract the KeySecure client tarball in the Key HSM library directory (`/usr/share/keytrustee-server-keyhsm/`).

- **Thales**

  Install the Thales client service. Copy `nCipherKM.jar`, `jcetools.jar`, and `rsaprivenc.jar` from the installation media (usually located in `opt/nfast/java/classes` relative to the installation media mount point) to the Key HSM library directory (`/usr/share/keytrustee-server-keyhsm/`).

See your HSM product documentation for more information on installing and configuring your HSM and client libraries.

> **Note:** When using an HSM with Key Trustee Server and Navigator Encrypt, encrypting a large number of directories may exceed the capacity of the HSM. For example, encrypting MapReduce spill files requires encrypting each HDFS data directory or disk on each node, each with its own encryption key. On a 10-node cluster with 12 disks per node, this requires 120 keys. Make sure that your HSM can support your encryption requirements.

To initialize Key HSM, use the `service keyhsm setup` command in conjunction with the name of the target HSM distribution:

```
$ sudo service keyhsm setup [keysecure|thales|luna]
```

For all HSM distributions, this first prompts for the IP address and port number that Key HSM listens on.

> **Important:** If you have implemented Key Trustee Server high availability, initialize Key HSM on each Key Trustee Server.

Cloudera recommends using the loopback address (`127.0.0.1`) for the listener IP address and `9090` as the port number.

If the setup utility successfully validates the listener IP address and port, you are prompted for additional information specific to your HSM. For HSM-specific instructions, continue to the HSM-Specific Setup for Cloudera Navigator Key HSM on page 302 section for your HSM.

The Key HSM keystore defaults to a strong, randomly-generated password. However, you can change the keystore password in the `application.properties` file:

```
keyhsm.keystore.password.set=yes
```

Then, run the `service keyhsm setup` command with the name of the HSM to which the keystore password applies. You will be prompted to enter the new keystore password, which must be a minimum of six characters in length:

```
$ sudo service keyhsm setup [keysecure|thales|luna]
```

After initial setup, configuration is stored in the `/usr/share/keytrustee-server-keyhsm/application.properties` file, which contains human-readable configuration information for the Navigator Key HSM server.

HSM-Specific Setup for Cloudera Navigator Key HSM

SafeNet KeySecure

Note: KeySecure was previously named DataSecure, but the Key HSM configuration process is the same for both.

After entering the Key HSM listener IP address and port, the HSM setup for SafeNet KeySecure prompts for login credentials, the IP address of the KeySecure HSM, and the port number:

```
-- Ingrian HSM Credential Configuration --
Please enter HSM login USERNAME: keyhsm
Please enter HSM login PASSWORD: ******

Please enter HSM IP Address or Hostname: 172.19.1.135
Please enter HSM Port number: 9020
```

If the connection is successful, the following message is displayed:

```
Valid address:                                    :[ Successful ]
```

The KeySecure setup utility then prompts you whether to use SSL:

```
Use SSL? [Y/n] Y
```

If you choose to use SSL, Key HSM attempts to resolve the server certificate, and prompts you to trust the certificate:

```
  [0]         Version: 3
         SerialNumber: 0
             IssuerDN: C=US,ST=TX,L=Austin,O=ACME,OU=Dev,
CN=172.19.1.135,E=webadmin@example.com
           Start Date: Thu Jan 29 09:55:57 EST 2015
           Final Date: Sat Jan 30 09:55:57 EST 2016
            SubjectDN: C=US,ST=TX,L=Austin,O=ACME,OU=Dev,
CN=172.19.1.135,E=webadmin@example.com
           Public Key: RSA Public Key
            modulus: abe4a8dcef92e145984309bd466b33b35562c7f875
                     1d1c406b1140e0584890272090424eb347647ba04b
                     34757cacc79652791427d0d8580a652c106bd26945
                     384b30b8107f8e15d2deba8a4e868bf17bb0207383
                     7cffef0ef16d5b5da5cfb4d3625c0affbda6320daf
                     7c6b6d8adfcb563960fcd1207c059300feb6513408
                     79dd2d929a5b986517531be93f113c8db780c92ddf
                     30f5c8bf2b0bea60359b67be306c520358cc0c3fc3
                     65500d8abeeac99e53cc2b369b2031174e72e6fca1
                     f9a4639e09240ed6d4a73073885868e814839b09d5
                     6aa98a5a1e230b46cdb4818321f546ac15567c5968
```

```
                     33be47ef156a73e537fd09605482790714f4a276e5
                     f126f935
    public exponent: 10001

  Signature Algorithm: SHA256WithRSAEncryption
            Signature: 235168c68567b27a30b14ab443388039ff12357f
                       99ba439c6214e4529120d6ccb4a9b95ab25f81b4
                       7deb9354608df45525184e75e80eb0948eae3e15
                       c25c1d58c4f86cb9616dc5c68dfe35f718a0b6b5
                       56f520317eb5b96b30cd9d027a0e42f60de6dd24
                       5598d1fcea262b405266f484143a74274922884e
                       362192c4f6417643da2df6dd1a538d6d5921e78e
                       20a14e29ca1bb82b57c02000fa4907bd9f3c890a
                       bdae380c0b4dc68710deeaef41576c0f767879a7
                       90f30a4b64a6afb3a1ace0f3ced17ae142ee6f18
                       5eff64e8b710606b28563dd99e8367a0d3cbab33
                       2e59c03cadce3a5f4e0aaa9d9165e96d062018f3
                       6a7e8e3075c40a95d61ebc8db43d77e7
       Extensions:
                       critical(false) BasicConstraints: isCa(true)
                       critical(false) NetscapeCertType: 0xc0

Trust this server? [y/N] Y

Trusted server:                                   :[ Successful ]
```

Thales HSM

By default, the Thales HSM client process listens on ports 9000 and 9001. The Cloudera Manager agent also listens on port 9000. To prevent a port conflict, you must change the Thales client ports. Cloudera recommends using ports 11500 and 11501.

To change the Thales client ports, run the following commands:

```
$ sudo /opt/nfast/bin/config-serverstartup --enable-tcp --enable-privileged-tcp
--port=11500 --privport=11501
$ sudo /opt/nfast/sbin/init.d-ncipher restart
```

To configure Key HSM to use the modified port, edit the `/usr/share/keytrustee-server-keyhsm/start.sh` file and add the `-DNFAST_SERVER_PORT=11500` Java system property. For example:

```
java -classpath "*:/usr/safenet/lunaclient/jsp/lib/*:/opt/nfast/java/classes/*"
-Djava.library.path=/usr/safenet/lunaclient/jsp/lib/ -DNFAST_SERVER_PORT=11500
com.cloudera.app.run.Program $@
```

Before completing the Thales HSM setup, run the `nfkminfo` command to verify that the Thales HSM is properly configured:

```
$ sudo /opt/nfast/bin/nfkminfo
  World generation  2
  state       0x17270000 Initialised Usable Recovery !PINRecovery !ExistingClient
              RTC  NVRAM FTO !AlwaysUseStrongPrimes SEEDebug
```

If `state` reports `!Usable` instead of `Usable`, configure the Thales HSM before continuing. See the Thales product documentation for instructions.

After entering the Key HSM listener IP address and port, the HSM setup for Thales prompts for the OCS card password:

```
Please enter the OCS Card Password (input suppressed):

Configuration saved in 'application.properties' file
Configuration stored in: 'application.properties'. (Note: You can also use service keyHsm
 settings to quickly view your current configuration)
```

Luna HSM

> **Important:** If you have implemented Key Trustee Server high availability, ensure that the Luna client on each Key Trustee Server is configured with access to the same partition. See the Luna product documentation for instructions on configuring the Luna client.

Before completing the Luna HSM setup, run the `vtl verify` command (usually located at `/usr/safenet/lunaclient/bin/vtl`) to verify that the Luna HSM is properly configured.

After entering the Key HSM listener IP address and port, the HSM setup for Luna prompts for the slot number and password:

```
-- Configuring SafeNet Luna HSM --
Please enter SafeNetHSM Slot Number: 1
Please enter SafeNet HSM password (input suppressed):
Configuration stored in: 'application.properties'. (Note: You can also use service keyHsm
 settings to quickly view your current configuration)
Configuration saved in 'application.properties' file
```

See the Luna product documentation for instructions on configuring your Luna HSM if you do not know what values to enter here.

Validating Key HSM Settings

After the setup completes, the Key HSM configuration is stored in `/usr/share/keytrustee-server-keyhsm/application.properties`.

You can view these settings using the `service keyhsm settings` command:

```
$ sudo service keyhsm settings

  # keyHsm Server Configuration information:
  keyhsm.management.address : 172.19.1.2
  keyhsm.server.port : 9090
  keyhsm.management.port : 9899
  keyhsm.service.port : 19791
  keyhsm.hardware : ncipher

  # Module OCS Password
  thales.ocs_password :
    GIqhXDuZsj1Oet137Lb+f+tqkYvKYDm/8StefpNqZWwlB+LfSYlB4eHd
    endtYJio8qLjjbT+e7j2th5xf8O9t8FwfVguuyFW+6wdD
    uNGvse1LY/itCwqF0ScMlB1Mnz4010xqC6ylPW7l+0JjjkkqqM5gJJbl8lsQFFaIGVM/pY=
```

These settings can be manually configured by modifying the `application.properties` file, with the exception of any passwords. These are encrypted by design, and can only be changed by re-running the setup utility.

Verifying Key HSM Connectivity to HSM

To verify Hardware Security Module (HSM) operations using Key HSM, run the following command on the Key Trustee Server host (which should also be the Key HSM host as described in Installing Cloudera Navigator Key HSM):

```
$ curl -k https://keytrustee01.example.com:11371/test_hsm
```

If Key HSM operations to the HSM are successful, the command returns output similar to the following:

```
"Sample Key TEST_HELLO_DEPOSIT2016-06-03-072718 has been created"
```

You must run this command from the Key Trustee Server host. If you run it from a different host, the command returns an HTTP 403 error code.

If the command returns an HTTP 405 error code, restart Key Trustee Server and try again.

> **Note:** If you are using the `test_hsm` script to verify that the Key Hardware Security Module (Key HSM) has successfully integrated with the Key Trustee Server, or to verify that the Key HSM is connected to HSM, *and* the Key Trustee Server private key file is password-protected, then the verification may fail. This can occur even if the integration is successful or connected.
>
> If this occurs, then create a key through Hadoop for the test.

Creating a Key Store with CA-Signed Certificate

Required Files

Before proceeding, ensure that you have the following three PEM files:

- Certificate Authority (CA) PEM file
- Signed PEM certificate
- Private key PEM file

The following example uses `ssl-cert-keyhsm-ca.pem`, `ssl-cert-keyhsm.pem`, and `ssl-cert-keyhsm-pk.pem`, respectively, to represent these files.

Create the Key Store

The following command accepts the `ssl-cert-keyhsm.pem` and `ssl-cert-keyhsm-pk.pem` files and converts them to a `.p12` file:

```
$ openssl pkcs12 -export -in ssl-cert-keyhsm.pem -inkey ssl-cert-keyhsm-pk.pem -out
mycert.p12 -name alias -CAfile ssl-cert-keyhsm-ca.pem -caname root -chain
```

> **Important:** The certificate CN must match the fully qualified domain name (FQDN) of the Key Trustee Server.

Managing the Navigator Key HSM Service

Use the `keyhsm` service for all basic server operations:

```
$ sudo service keyhsm
keyHsm service usage:
    setup <hsm name> - setup a new connection to an HSM
    trust <path>     - add a trusted client certificate
    validate         - validate that keyHSM is properly configured
    settings         - display the current server configuration
    start            - start the keyHSM proxy server
    status           - show the current keyHSM server status
    shutdown         - force keyHSM server to shut down
    reload           - reload the server (without shutdown)
```

The `reload` command causes the application to restart internal services without ending the process itself. If you want to stop and start the process, use the `restart` command.

Logging and Audits

The Navigator Key HSM logs contain all log and audit information, and by default are stored in the `/var/log/keyhsm` directory.

You can configure the maximum log size (in bytes) and maximum number of log files to retain by adding or editing the following entries in `/usr/share/keytrustee-server-keyhsm/application.properties`:

```
keyhsm.log.size = 100000000
keyhsm.roll.size = 3
```

The values used in this example are the default values, and are used if these parameters are not set.

To enable debug logging, add the debug parameter to the start command:

```
$ sudo service keyhsm start debug
```

Note: You cannot start Key HSM in debug mode using the `systemctl` command on RHEL 7-compatible OS. You must use the `service` command.

This enables debug logging until the service is restarted without the debug parameter.

Integrating Key HSM with Key Trustee Server

Using a hardware security module with Navigator Key Trustee Server requires Key HSM. This service functions as a driver to support interactions between Navigator Key Trustee Server and the hardware security module, and it must be installed on the same host system as Key Trustee Server. The steps below assume that both Key HSM and Key Trustee Server are set up and running. See Installing Cloudera Navigator Key HSM for details. Integrating Key HSM and Key Trustee Server involves the following steps:

1. Check Existing Key Names (for existing Key Trustee Server users only)
2. Establish Trust from Key HSM to Key Trustee Server
3. Integrate Key HSM and Key Trustee Server

Check Existing Key Names

During the process detailed below, you are prompted to migrate any existing keys from the Key Trustee Server to the HSM.

Warning: Migration fails if any existing keys do not adhere to the constraints.

Successful migration depends on the existing keys conforming to the following constraints:

- Key names can begin with alpha-numeric characters only
- Key names can include only these special characters:
  - Hyphen -
  - Period .
  - Underscore _

If any existing key names in Key Trustee Server do not meet the requirements listed above, the migration fails. To prepare for migration, check your key names and do the following if any of them are non-conforming:

- Decrypt any data using the non-conforming key
- Create a new key, named per the requirements
- Re-encrypt the data using the new key

After this, the migration from Key Trustee Server to the HSM will succeed during the process below.

Important: Keys are not available during migration, so you should perform these tasks during a maintenance window.

Establish Trust from Key HSM to Key Trustee Server

This step assumes that Key Trustee Server has a certificate for TLS (wire) encryption as detailed in Managing Key Trustee Server Certificates on page 298. Before you can run the commands in the steps below, Key HSM service must explicitly trust the Key Trustee Server certificate (presented during TLS handshake). To establish this trust, run the following command:

```
$ sudo keyhsm trust /path/to/key_trustee_server/cert
```

The `/path/to/key_trustee_server/cert` in this command (and in the commands below) depends on whether the Key Trustee Server uses the **default certificate** (created by default during install), or uses a **custom certificate** (obtained from a commercial or internal CA). The two alternate paths are shown in the table below. The custom path is a common example but may differ from that shown.

Default	Custom
/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keyfile.pem	/etc/pki/cloudera/certs/ *cert-file.crt*
/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-pk.pem	/etc/pki/cloudera/private/*private-key.key*

Note: The system requires TLS and Kerberos authentication throughout the system for security reasons. Connections attempted over SSL (1 through 3) and connections from untrusted clients are immediately terminated to prevent POODLE (Padding Oracle On Downgraded Legacy Encyryption) exploits. See the Cloudera Security Bulletin for more information.

Integrate Key HSM and Key Trustee Server

The steps below assume that both Key HSM and the Key Trustee Server are on the same host system, as detailed in Installing Cloudera Navigator Key HSM. These steps invoke commands on the Key HSM service and the Key Trustee Server, and they must be run on the host—they cannot be run remotely from another host.

1. Ensure the Key HSM service is running:

```
$ sudo service keyhsm start
```

2. Establish trust from Key Trustee Server to Key HSM specifying the path to the private key and certificate (Key Trustee Server is a client to Key HSM). This example shows how to use the `--client-certfile` and `--client-keyfile` options to specify the path to non-default certificate and key:

```
$ sudo ktadmin keyhsm --server https://keyhsm01.example.com:9090 \
--client-certfile  /etc/pki/cloudera/certs/mycert.crt \
--client-keyfile /etc/pki/cloudera/certs/mykey.key --trust
```

   For a password-protected Key Trustee Server private key, add the `--passphrase` argument to the command and enter the password when prompted:

```
$ sudo ktadmin keyhsm --passphrase \
--server https://keyhsm01.example.com:9090 \
--client-certfile  /etc/pki/cloudera/certs/mycert.crt \
--client-keyfile /etc/pki/cloudera/certs/mykey.key --trust
```

3. Restart Key Trustee Server:

- **Using Cloudera Manager:** Restart the Key Trustee Server service (**Key Trustee Server service** > **Actions** > **Restart**).
- **Using the Command Line:** Restart the Key Trustee Server daemon:
  - RHEL 6-compatible: `$ sudo service keytrusteed restart`
  - RHEL 7-compatible: `$ sudo systemctl restart keytrusteed`

4. Verify connectivity between the Key HSM service and the HSM:

```
$ curl -k https://keytrustee01.example.com:11371/test_hsm
```

> **Important:** You must perform the connection verification between Key HSM and the HSM for all Key Trustee Server hosts.

Successful connection and test of operations returns output like the following:

```
"Sample Key TEST_HELLO_DEPOSIT2016-06-03-072718 has been created"
```

> **Note:** If you are using the `test_hsm` script to verify that the Key Hardware Security Module (Key HSM) has successfully integrated with the Key Trustee Server, or to verify that the Key HSM is connected to HSM, *and* the Key Trustee Server private key file is password-protected, then the verification may fail. This can occur even if the integration is successful or connected.
>
> If this occurs, then create a key through Hadoop for the test.

See Verifying Key HSM Connectivity to HSM on page 304 for more information about the validation process.

Cloudera Navigator Encrypt

Cloudera Navigator Encrypt transparently encrypts and secures data at rest without requiring changes to your applications and ensures there is minimal performance lag in the encryption or decryption process. Advanced key management with Cloudera Navigator Key Trustee Server on page 283 and process-based access controls in Navigator Encrypt enable organizations to meet compliance regulations and ensure unauthorized parties or malicious actors never gain access to encrypted data.

For instructions on installing Navigator Encrypt, see Installing Cloudera Navigator Encrypt.

For instructions on configuring Navigator Encrypt, continue reading:

Registering Cloudera Navigator Encrypt with Key Trustee Server

Prerequisites

Functioning Navigator Key Trustee Server

After Installing Cloudera Navigator Encrypt on a host, you must register the host with Navigator Key Trustee Server. If you have not yet installed Navigator Key Trustee Server, see Installing Cloudera Navigator Key Trustee Server for instructions.

Key Trustee Server Organization

To register with Key Trustee Server, you must have an existing organization. See Managing Key Trustee Server Organizations on page 295 for information on creating and viewing organizations on a Key Trustee Server.

Master Password

The Master Key is the primary Navigator Encrypt administrator access code and is configured by the Navigator Encrypt administrator during installation. The Master Key can take any one of three different forms:

- If you choose a passphrase (single), it must be between 15 and 32 characters long.
- If you choose passphrase (dual), both must be between 15 and 32 characters long.
- If you choose the RSA option, enter a path to the RSA key file, and if it has RSA passphrase, enter it for this private key.

> **Warning:** It is ***extremely*** important that you keep your master password secret and safe. In the event that you lose your master password, **you will never be able to recover it**, leaving your encrypted data **irretrievably locked away**.

Registering with Key Trustee Server

After Installing Cloudera Navigator Encrypt on a host, you must register the host with Navigator Key Trustee Server to be able to encrypt and decrypt data. The following section lists the command options for registering your Navigator Encrypt client.

> **Note:** Do not run Navigator Encrypt commands with the `screen` utility.

Example command:

```
$ sudo navencrypt register --server=https://keytrustee01.example.com:11371
--passive-server=https://keytrustee02.example.com:11371 --org=your_keytrustee_org
--auth=org_auth_token
```

Table 26: Registration Options

Command Option	Explanation
`--clientname=my_client_name`	User-defined unique name for this client to be used for administration and reports. You can verify your client name in the `/etc/navencrypt/keytrustee/clientname` file.
`--server=https://keytrustee01.example.com:11371`	Target Active Key Trustee Server for key storage. Replace *keytrustee01.example.com:11371* with the hostname and port of the Active Key Trustee Server. The default port is `11371`.
`--passive-server=https://keytrustee02.example.com:11371`	Target Passive Key Trustee Server for key storage. Replace *keytrustee02.example.com:11371* with the hostname and port of the Passive Key Trustee Server. The default port is `11371`.
`--org=your_keytrustee_org`	Key Trustee organization name configured by the Key Trustee Server administrator
`--auth=org_auth_token`	Organization authorization token, a pre-shared secret by the Navigator Key Trustee Server administrator
`--skip-ssl-check`	Skip SSL certificate verification. Use with self-signed certificates on the Navigator Key Trustee Server
`--trustee`	Add trustees for retrieval of the master key
`--votes`	Configure voting policy for trustees
`--recoverable`	Master Key will be uploaded without encrypting it with your local GPG Navigator Key Trustee
`--scheme "<scheme>"`	Key Trustee Server scheme that Navigator Encrypt uses for public key operations. Specify `"http"` or `"https"`.
`--port`	Key Trustee Server port that Navigator Encrypt uses for public key operations.

Registering with Previous Versions of Key Trustee Server

By default, new installations of Navigator Key Trustee Server 5.4.0 use a single HTTPS port for key storage and public key operations. Previous versions and upgrades use separate ports for key storage and public key operations. For backward compatibility, Navigator Encrypt 3.7.0 introduces the `--scheme` and `--port` parameters for the `navencrypt register` command.

For example, to register a version 3.7.0 Navigator Encrypt client with a version 3.8.0 Key Trustee Server using HTTPS over port 443 for key storage and HTTP over port 80 for public key operations, run the following command:

```
$ sudo navencrypt register --server=https://keytrustee.example.com:443
--org=key_trustee_org --auth=auth_token --scheme "http" --port 80
```

Navigator Encrypt versions lower than 3.7.0 do not support the `--scheme` and `--port` parameters. For these versions of Navigator Encrypt, you must ensure that the Key Trustee Server is configured to use port 443 (HTTPS) for key storage and port 80 (HTTP) for public key operations.

Navigator Encrypt versions lower than 3.8.0 do not support the `--passive-server` parameter.

Updating Key Trustee Server Ports

The `navencrypt register` command does not provide the ability to change the ports for existing registrations. If the Key Trustee Server ports are changed, you must update `/etc/navencrypt/keytrustee/ztrustee.conf` with the new port and scheme parameters (HKP_PORT and HKP_SCHEME, respectively).

For example, see the following `ztrustee.conf` excerpt from a registered client that has been upgraded to Navigator Encrypt 3.7.0:

```
{
        "LOCAL_FINGERPRINT":    "2048R/182AAA838DC300AC334258D8E7F299BFB68A6F6F",
        "REMOTES":      {
                "kts01.example.com": {
                        "REMOTE_FINGERPRINT":
"4096R/AF6400E12DC149799CA8CE6BF1604C34D830DE20",
                        "REMOTE_SERVER":        "https://kts01.example.com",
                        "DEFAULT":      true,
                        "SSL_INSECURE": false,
                        "PROTOCOL":     "json-encrypt"
                }
        }
}
```

In this example, the Key Trustee Server (`keytrustee.example.com`) is using the default configuration of port 443 (HTTPS) for key storage and port 80 (HTTP) for public key operations.

If the Key Trustee Server is then updated to use port 11371 (HTTPS) for both key storage and public key operations, you must update `ztrustee.conf` as follows (changes in **bold**):

```
{
        "LOCAL_FINGERPRINT":    "2048R/182AAA838DC300AC334258D8E7F299BFB68A6F6F",
        "REMOTES":      {
                "kts01.example.com": {
                        "REMOTE_FINGERPRINT":
"4096R/AF6400E12DC149799CA8CE6BF1604C34D830DE20",
                        "REMOTE_SERVER":        "https://kts01.example.com:11371",
                        "HKP_PORT":     11371,
                        "HKP_SCHEME":   "https",
                        "DEFAULT":      true,
                        "SSL_INSECURE": false,
                        "PROTOCOL":     "json-encrypt"
                }
        }
}
```

Updating Navigator Encrypt for High Availability Key Trustee Server

If you registered a Navigator Encrypt client with a standalone Key Trustee Server, and then configured high availability for Key Trustee Server, you can edit `/etc/navencrypt/keytrustee/ztrustee.conf` to enable the client to take advantage of the high availability features. The following example shows the contents of `ztrustee.conf` after adding the required REMOTE_SERVERS entry (changes in **bold**):

```
{
        "LOCAL_FINGERPRINT":    "2048R/182AAA838DC300AC334258D8E7F299BFB68A6F6F",
        "REMOTES":      {
                "kts01.example.com": {
                        "REMOTE_FINGERPRINT":
"4096R/AF6400E12DC149799CA8CE6BF1604C34D830DE20",
                        "REMOTE_SERVER":        "https://kts01.example.com:11371",
                        "HKP_PORT":     11371,
```

```
                        "HKP_SCHEME":   "https",
                        "DEFAULT":      true,
                        "REMOTE_SERVERS":       ["https://kts01.example.com:11371",
"https://kts02.example.com:11371"],
                        "SSL_INSECURE": true,
                        "PROTOCOL":     "json-encrypt"
                }
        }
}
```

Configuration Files

The installer creates the `/etc/navencrypt` directory. All configuration settings are saved in this directory. **Do not** delete any file from `/etc/navencrypt`. These files provide the necessary information for the Navigator Encrypt application to function properly.

> **Warning:** Perform backups of encrypted data, mount-points, and Navigator Encrypt configuration directories on a regular basis. To do this, ensure you have a backup of `/etc/navencrypt`. **Failure to backup this directory will make your backed up encrypted data unrecoverable in the event of data loss.**

Change Master Key by UUID

It is possible to re-use a previously used Master Key by its UUID. For example, if you currently have a Master key with a single passphrase, you can see the corresponding Navigator Key Trustee UUID in the `/etc/navencrypt/control` file:

```
$ cat /etc/navencrypt/control
{
 "app": {
  "name": "navencrypt",
  "version": "3.5"
 },
 "keys": {
  "master": {
   "type": "single-passphrase",
   "uuid": "qMAKRMdk4HVbhfzR79cp9w92YBmNHJ5nSLhfd8ZVo6L"
  },
  "targets": []
 }
}
```

> **Note:** If the control file is accidentally deleted, you can restore it using the `navencrypt control --restore-control-file` command.

You can copy the UUID (`qMAKRMdk4HVbhfzR79cp9w92YBmNHJ5nSLhfd8ZVo6L` in this example) and run `navencrypt key --change` with option `--new-master-key-uuid` to change a Master Key by using its UUID only:

```
$ sudo navencrypt key --change
--new-master-key-uuid=qMAKRMdk4HVbhfzR79cp9w92YBmNHJ5nSLhfd8ZVo6L
>> Type your OLD Master key
Type MASTER passphrase 1:
Type MASTER passphrase 2:
Verifying Master Key against Navigator Key Trustee (wait a moment)...
OK
Changing Master key (wait a moment)...
 * Setting up EXISTING MASTER key...
 * Uploading CONTROL content...
 * Re-encrypting local keys...
Master key successfully changed.
```

> **Note:** The `navencrypt key` command fails if no volumes are encrypted or the kernel module is not loaded.

Preparing for Encryption Using Cloudera Navigator Encrypt

Before you can encrypt data, you must prepare a storage repository to hold the encrypted data and a mount point through which to access the encrypted data. The storage repository and mount point must exist before encrypting data using the `navencrypt-move` command.

Data stored and retrieved from the repository is encrypted and decrypted transparently.

Cloudera Navigator Encrypt *does not* support:

- Encrypting a directory that contains or is contained within a mount point for another service (including Navigator Encrypt and NFS). See Encrypting Data on page 319 for more information.
- Encrypting immutable files or directories containing immutable files.
- Installation or use in `chroot` environments, including creating `chroot` environments within an encrypted directory.

Navigator Encrypt Commands

> **Note:** Do not run Navigator Encrypt commands with the `screen` utility.

The following table lists the commands used to encrypt data:

Table 27: Navigator Encrypt Commands

Command	Description
`navencrypt`	Manage, update, and verify your data.
`navencrypt-prepare`	Prepare your system for encryption by creating mount-points and specifying storage.
`navencrypt-prepare --undo`	Remove a mountpoint that is no longer in use.
`navencrypt-move`	Encrypt/decrypt your data to/from the encrypted filesystem.
`navencrypt-profile`	Generate process profile information in JSON format.
`navencrypt-module-setup`	Build or rebuild the Navigator Encrypt kernel module.

Preparing for Encryption

> **Note:** When using an HSM with Key Trustee Server and Navigator Encrypt, encrypting a large number of directories may exceed the capacity of the HSM. For example, encrypting MapReduce spill files requires encrypting each HDFS data directory or disk on each node, each with its own encryption key. On a 10-node cluster with 12 disks per node, this requires 120 keys. Make sure that your HSM can support your encryption requirements.

To get an in-depth look at the details behind the `navencrypt-prepare` command, or to use a unique configuration, use the interactive prompt by executing `navencrypt-prepare` with no options. This launches an interactive console that guides you through the following operations:

- Creating internal encryption keys

- Registering internal keys in Navigator Key Trustee
- Registering mount point in `/etc/navencrypt/ztab`
- Mounting current mount point
- Establishing encryption method (`dm-crypt` for devices, `ecryptfs` for directories)

Using the console, you can choose how you want your data stored and accessed. Navigator Encrypt offers two different types of encryption:

- Block-level encryption with `dm-crypt`: Protect your data by encrypting the entire device. This option enables full disk encryption and is optimized for some system configurations. Block-level encryption can be used with logical devices such as a loop device.
- File-level encryption with `ecryptfs`: Protect your data by mounting an encrypted filesystem on top of an existing one. Enables transparent access to encrypted data without modifying your storage.

> **Note:** As of August 2015, Filesystem-level encryption using eCryptfs is deprecated. For new installations, use block-level encryption. For existing installations using eCryptfs, see Migrating eCryptfs-Encrypted Data to dm-crypt on page 321 for instructions on migrating data encrypted using eCryptfs to use dm-crypt.

See Block-Level Encryption with dm-crypt on page 315 and Filesystem-Level Encryption with eCryptfs on page 317 for more information.

In order to prepare for encryption, you must set a location to store the encrypted data and a mount point through which to access the data. The storage location and mount point must be created before encrypting data.

In the following example, we will use the directory `/navencrypt/encrypted-storage` for the encrypted storage and `/navencrypt/mount-point` for the mount point. If you have specific space/partition requirements, you can select a different directory, although Cloudera highly recommends that you place the encrypted directory on the same partition as the data you are planning to encrypt.

The syntax for the prepare command is as follows:

```
$ sudo navencrypt-prepare <data_storage_directory> <mount_point>
```

When specifying the storage path and the mount point path, *do not* use a trailing / in the path names. Both directories must exist prior to running the `navencrypt-prepare` command. They are not automatically created.

To create the encrypted partition, create the mount point and storage directories, and then use the `navencrypt-prepare` utility:

```
$ sudo mkdir -p /navencrypt/encrypted-storage /navencrypt/mount-point
$ sudo navencrypt-prepare /navencrypt/encrypted-storage /navencrypt/mount-point
```

For RHEL 7, run the following command after the `navencrypt-prepare` command completes:

```
$ sudo systemctl start navencrypt-mount
```

To demonstrate the difference between the two directories, this example uses different directories for the encrypted storage and the mount point. It is also possible to store and access the data using the same directory.

To see the effects of these commands, run `df -h`. This command displays the partition information about your system. You should see an `ecryptfs` partition located at `/navencrypt/encrypted-storage`, and mounted at `/navencrypt/mount-point`.

After you have successfully prepared a client for encryption, you can encrypt and decrypt data using the commands described in Encrypting and Decrypting Data Using Cloudera Navigator Encrypt on page 319.

Navigator Encrypt and Device UUIDs

Navigator Encrypt has always prepared and identified devices simply using a device name, such as `/dev/sdb1` or `/dev/loop0`. However, we know that using a device name or label could lead to a conflict and impact system operations.

Navigator Encrypt also supports preparing devices using a UUID, in addition to device name. This UUID is simply a symbolic link to the actual device, and is created when preparing a device with Navigator Encrypt during a `navencrypt-prepare` operation.

The advantage of using a device UUID is that if a device's name changes, the UUID associated with that device does *not* change. To ensure that Navigator Encrypt recognizes devices even when the device name changes, enter the command:

```
navencrypt-prepare --use-uuid /dev/sda1 /mountpoint
```

To unprepare (ensure the device UUID is included), enter either of the following commands:

```
navencrypt-prepare --undo-force /dev/disk/by-uuid/3a602a15-11f7-46ac-ae98-0a51e1b25cf9
navencrypt-prepare --undo /dev/disk/by-uuid/3a602a15-11f7-46ac-ae98-0a51e1b25cf9
```

Note: While the device name is still used in the `navencrypt-prepare` statement, rest assured that Navigator Encrypt handles the device by the UUID, which is calculated and used for mount during boot. The device name is used for convenience so that you do not have to explicitly input the UUID in the command. Ultimately, Navigator Encrypt handles the device via the device UUID rather than the device name.

Important: UUID device support does *not* include loop devices; rather, it only applies to physical devices. When preparing loop devices with Navigator Encrypt, always use the device name.

Block-Level Encryption with dm-crypt

Note: For best performance, Cloudera strongly recommends using block encryption with dm-crypt. See Migrating eCryptfs-Encrypted Data to dm-crypt on page 321 for instructions on migrating data encrypted using eCryptfs to use dm-crypt.

When choosing block-level encryption during the interactive console, you must specify two parameters:

1. The first parameter is the block device that you want to store the encrypted file system in. Because this device stores all of the encrypted data, it must be as large as or larger than the target data. The device must exist and be empty. Supported storage devices are:
   - Physical block devices (for example, a disk device)
   - Virtual block devices (for example, a block device created by LVM)
   - Loop devices (see Block-Level Encryption with a Loop Device on page 316 for instructions on creating a loop device)

   Important: If the block device to be used for encryption was previously used by the host, entries for it must be removed from the file `/etc/fstab` before running the `navencrypt-prepare` command.

2. The second parameter is the mount point for the encrypted file system. This is the location where you can access the encrypted data stored in the first parameter. The mount point must already exist. It is not created by the `navencrypt-prepare` command.

The entire device in the first parameter is used for encrypted data.

> **Note:** Do not manually unmount the encryption mount point (for example, using `umount`). If you do so, you must manually close the `dm-crypt` device using the following procedure:
>
> 1. Run `dmsetup table` to list the `dm-crypt` devices.
> 2. Run `cryptsetup luksClose <device_name>` to close the device for the unmounted mount point.

After choosing these two parameters and following the interactive console (discussed further in Preparing for Encryption on page 313), you are ready to encrypt your data. The following example shows successful output from a `navencrypt-prepare` command using dm-crypt for block-level encryption:

```
$ sudo /usr/sbin/navencrypt-prepare /dev/sda1 /mnt/dm_encrypted
Type MASTER passphrase:
Encryption Type:  dmCrypt (LUKS)
Cipher:          aes
Key Size:  256
Random Interface: /dev/urandom
Filesystem:    ext4
Verifying MASTER key against Navigator Key Trustee (wait a moment) ... OK
Generation Encryption Keys with /dev/urandom        ... OK
Preparing dmCrypt device (--use-urandom)                ... OK
Creating ext4 filesystem                                ... OK
Registering Encryption Keys (wait a moment)         ... OK
Mounting /dev/sda1                                    ... OK
```

Block-Level Encryption with a Loop Device

A block-level encrypted device can be a physical device or a storage space treated as a device. See Migrating eCryptfs-Encrypted Data to dm-crypt on page 321 for instructions on migrating data encrypted using eCryptfs to use dm-crypt with a loop device.

To configure a loop device, use the dd command to create a storage space:

> **Warning:** The space for the loop device is pre-allocated. After the loop device is created, the size cannot be changed. Make sure you are allocating enough storage for the current encrypted data as well as any future data.
>
> If your disks are mounted individually with a filesystem on each disk, and your storage needs exceed the capacity of a single disk, you can create a loop device on each disk for which want to allocate space for encrypted data. If you have consolidated storage (for example, using LVM), you can create a single loop device or multiple devices.

```
$ sudo dd if=/dev/zero of=/dmcrypt/storage bs=1G count=500
```

The dd command above creates a 500 GB file. Modify the `bs` and `count` values to generate the required file size.

After generating the file, run `losetup -f` to view unused loop devices. Use the available loop device with the `navencrypt-prepare -d` command, demonstrated below.

Specifically for loop devices, the `-d` parameter enables Navigator Encrypt to manage the loop device association. You no longer need to use the `losetup` command to associate the file with the loop device, and the loop device is automatically prepared at boot. For RHEL 7-compatible OS, you must run the following commands to ensure that a loop device is available at boot:

```
$ sudo bash -c 'echo "loop" > /etc/modules-load.d/loop.conf'
$ sudo bash -c 'echo "options loop max_loop=8" > /etc/modprobe.d/loop_options.conf'
```

> **Warning:** Loop devices are not created by Navigator Encrypt. Instead, Navigator Encrypt assigns a datastore to a loop device when the `navencrypt-prepare --datastore` option is used. So, it is up to the system administrator to create persistent /dev/loopX devices, which are required to prepare a virtual block device. If the loop device being prepared is not persistent, then Navigator Encrypt will not mount the device upon a reboot.

The data storage directory name (`/dmcrypt/storage` in the previous example) must contain only alphanumeric characters, spaces, hyphens (-), or underscores (_). Other special characters are not supported.

The following example shows the output from a successful command:

```
$ losetup -f
/dev/loop0
$ sudo navencrypt-prepare -d /dmcrypt/storage /dev/loop0 /dmcrypt/mountpoint
Type MASTER passphrase:

Encryption Type:  dmCrypt (LUKS)
Cipher:           aes
Key Size:         256
Random Interface: OpenSSL
Filesystem:       ext4
Options:

Verifying MASTER key against KeyTrustee (wait a moment)   ... OK
Generation Encryption Keys with OpenSSL                   ... OK
Assigning '/dev/loop0'->'/dmcrypt/storage'                ... OK
Preparing dmCrypt device                                  ... OK
Creating ext4 filesystem                                  ... OK
Registering Encryption Keys (wait a moment)               ... OK
Mounting /dev/loop0                                       ... OK
```

For upgraded Navigator Encrypt clients that already use loop devices, you can enable Navigator Encrypt to manage the loop device file association (instead of configuring the system to run the `losetup` command at boot) by adding the `nav_datastore` option to the entry in `/etc/navencrypt/ztab`. For example:

```
# <target mount-dir>        <source device>      <type>    <options>
/dmcrypt/mountpoint      /dev/loop0       luks
key=keytrustee,nav_datastore='/dmcrypt/storage'
```

> **Important:** Use caution when editing the `/etc/navencrypt/ztab` file. Entries are tab-separated (not space-separated). The `ztab` file must not contain empty lines.

After you have created the loop device, continue with the instructions in Block-Level Encryption with dm-crypt on page 315.

Filesystem-Level Encryption with eCryptfs

> **Note:** As of August 2015, Filesystem-level encryption using eCryptfs is deprecated. For best performance, Cloudera strongly recommends using Block-Level Encryption with dm-crypt on page 315 where possible. See Migrating eCryptfs-Encrypted Data to dm-crypt on page 321 for instructions on migrating data encrypted using eCryptfs to use dm-crypt.
>
> RHEL 7 does not support eCryptfs. For new installations on RHEL 7, you must use Block-Level Encryption with dm-crypt on page 315. If you are planning on upgrading to RHEL 7 and are currently using eCryptfs, migrate to dm-crypt before upgrading.

When choosing file-level encryption during the interactive console, you must specify two parameters:

1. The first parameter is the storage directory you want to store the encrypted file system in. Because this directory will hold all of the encrypted data, it must be as large as or larger than the target data.

2. The second parameter is the mount point for the encrypted file system. This is the location where you can access the encrypted data stored in the location identified by the first parameter.

While the data is technically stored at the location identified by the first parameter, you can only access the data from the mount point identified by the second parameter. Consider this when choosing where to mount your data.

After choosing these two parameters and following the interactive console (discussed further in Preparing for Encryption on page 313), you are ready to encrypt your data.

Undo Operation

Navigator Encrypt 3.5 and higher supports a new command option, `navencrypt-prepare --undo`. This command reverses the operations from the regular `navencrypt-prepare` command by removing the device from Navigator Encrypt control and removing registered encryption keys.

The only parameter of the undo operation is the storage device used to store the encrypted file system (not the mount point). Here is an example showing `navencrypt-prepare` and `navencrypt-prepare --undo` operations:

```
$ sudo navencrypt-prepare /path/to/storage /path/to/mountpoint
Type MASTER passphrase:

Encryption Type:  eCryptfs
Cipher:           aes
Key Size:         256
Random Interface: OpenSSL
Filesystem:       ext4
Options:

Verifying MASTER key against Navigator Key Trustee (wait a moment)     ... OK
Generation Encryption Keys with OpenSSL                        ... OK
Registering Encryption Keys (wait a moment)                    ... OK
Mounting /path/to/mountpoint                                   ... OK
$ sudo navencrypt-prepare --undo /path/to/storage
Type MASTER passphrase:
Verifying MASTER key against Navigator Key Trustee (wait a moment)     ... OK
Umounting /path/to/mountpoint                                  ... OK
```

Pass-through Mount Options for `navencrypt-prepare`

Navigator Encrypt 3.5 and higher provides the ability to specify options to pass to the `mount` command that is executed during `/etc/init.d/navencrypt-mount start` (`systemctl start navencrypt-mount` on RHEL 7). These options are specified with the `-o` option when preparing a mountpoint with the `navencrypt-prepare` command.

The following shows an example `navencrypt-prepare` command output when passing mount options with the `-o` option:

```
$ sudo navencrypt-prepare -o discard,resize /mnt/t2 /mnt/t2
Type MASTER passphrase:

Encryption Type:  eCryptfs
Cipher:           aes
Key Size:         256
Random Interface: OpenSSL
Filesystem:       ext4
Options:          discard,resize

Verifying MASTER key against Navigator Key Trustee(wait a moment)     ... OK
Generation Encryption Keys with OpenSSL                               ... OK
Registering Encryption Keys (wait a moment)                           ... OK
Mounting /mnt/t2                                                      ... OK
```

You can verify the results by viewing the `/etc/navencrypt/ztab` file:

```
$ cat /etc/navencrypt/ztab
/mnt/t2 /mnt/t2 ecryptfs key=keytrustee,cipher=aes,keysize=256,discard,resize
```

Options can be added or removed to existing mount points prepared with versions of Navigator Encrypt prior to 3.5 by editing the `/etc/navencrypt/ztab` file and adding the comma-separated options (no spaces) to the end of each line as seen in the previous example above.

> **Important:** Use caution when editing the `/etc/navencrypt/ztab` file. Entries are tab-separated (not space-separated). The `ztab` file must not contain empty lines.

To see the mounted filesystems and options, run `mount`:

```
$ mount
/mnt/t2 on /mnt/t2 type ecryptfs
(rw,ecryptfs_sig=6de3db1e87077adb,ecryptfs_unlink_sigs,noauto,\
ecryptfs_cipher=aes,ecryptfs_key_bytes=32,discard,resize)
```

Pass-through mount options work for both dm-crypt and eCryptfs. For a list of available mount options, see the man pages for `cryptsetup` and `ecryptfs` respectively.

Encrypting and Decrypting Data Using Cloudera Navigator Encrypt

> **Warning:** Before encrypting or decrypting any data, stop all processes (for example, MySQL, MongoDB, PostgreSQL, and so on) that have access to the target data. **Failure to do so could lead to data corruption.**

After the encrypted file system is created and initialized, it is ready to hold data. All encryption and decryption functionality is performed with a single command: `navencrypt-move`.

Do not manually create directories or files under a Cloudera Navigator Encrypt mount point; use only the `navencrypt-move` command to encrypt and decrypt data. See Preparing for Encryption Using Cloudera Navigator Encrypt on page 313 for more information about mount points.

After encrypting a file or directory, all data written and read through the mount point is transparently encrypted and decrypted.

Before You Begin

Navigator Encrypt does not support encrypting data in certain environments, including the following:

- *Do not* attempt to encrypt a directory that contains or is contained within a mount point for another service (including Navigator Encrypt and NFS). For example:
  - If your encryption mount point is `/var/lib/navencrypt/mount`, do not attempt to encrypt `/var`, `/var/lib`, `/var/lib/navencrypt`, `/var/lib/navencrypt/mount`, or anything under `/var/lib/navencrypt/mount/`.
  - If you have mounted an NFS filesystem at `/mnt/home`, do not attempt to encrypt `/mnt`, `/mnt/home`, or anything under `/mnt/home`.
- Do not attempt to encrypt immutable files or directories containing immutable files.
- Do not use Navigator Encrypt within a `chroot` environment, or create a `chroot` environment within an encrypted directory.
- If your Key Trustee Server is managed by Cloudera Manager, *do not* encrypt the Cloudera Manager database with Navigator Encrypt; doing so prevents Cloudera Manager from starting.

Encrypting Data

Do not manually create directories or files under a Navigator Encrypt mount point; use only the `navencrypt-move` command to encrypt data.

Here is an example command to encrypt data, with an explanation for each option:

```
$ sudo navencrypt-move encrypt @<category> <directory_or_file_to_encrypt>
<encrypted_mount_point>
```

Important: *Do not* run `navencrypt-move` commands simultaneously in multiple terminals. Doing so results in failure to encrypt or decrypt all of the specified data. No data is lost, as the source data is not removed, but you must re-run the failed operations sequentially.

Table 28: `navencrypt-move` Command Options

Command Option	Explanation
`navencrypt-move`	Main command interface for all actions that require moving data either to or from the encrypted file system. For more information see the `navencrypt-move` man page (`man navencrypt-move`).
`encrypt`	Identifies the cryptographic operation, in this case, encrypting data. The `decrypt` option is described later in Decrypting Data on page 321. **Note:** By default, all Navigator Encrypt encryption commands require free space equal to twice the size of the encrypted data. If your environment does not have enough free space, add `--per-file` to the end of the command. This moves each file individually. Per-file encryption only requires free space equal to twice the size of the largest individual file, but is a slower operation.
`@<category>`	The access category that is applied to the data being encrypted. Encrypted data is protected by process-based access controls that restrict access to only the processes that you allow. You can use any naming convention you want (the `@` symbol is required), but Cloudera recommends keeping it simple and memorable. For example, you can use a name referencing the data type being encrypted, such as `@mysql` for a MySQL deployment. See Listing Categories on page 330 for instructions on viewing existing categories.
`<directory_or_file_to_encrypt>`	The data that you want to encrypt. This can be a single file or an entire directory. Navigator Encrypt starts after the system boots, so do not encrypt required system files and directories (such as the root partition, `/var`, and so on). Some examples of recommended data directories to encrypt are `/var/lib/mysql/data`, `/db/data`, and so on.
`<encrypted_mount_point>`	Where you want to store the data. This is the path to the mount point specified during the `navencrypt-prepare` command.

When a file is encrypted, a symbolic link (symlink) is created which points to a mount point `@<category>` directory. The `navencrypt-move` command moves all specified data to the encrypted filesystem and replaces it with a symlink to the mount point for that encrypted filesystem.

Encrypting a directory is similar to encrypting a file. The following command encrypts a directory:

```
$ sudo /usr/sbin/navencrypt-move encrypt @mycategory /path/to/directory_to_encrypt/
/path/to/mount
```

In this command, a directory is specified instead of a filename, and a symlink is created for that particular directory. To see the effects of this command, run:

```
$ ls -l <directory_to_encrypt>
$ du -h <encrypted_storage_directory>
```

The output demonstrates the new filesystem layout. Everything that was in the target directory is now securely stored in the encrypted filesystem.

Decrypting Data

The decryption command requires only the path to the original data, which is now a symbolic link, as an argument. The following example demonstrates how to decrypt a file using the `navencrypt-move` command:

```
$ sudo /usr/sbin/navencrypt-move decrypt /path/to/encrypted/directory_or_file
```

> **Important:** *Do not* run `navencrypt-move` commands simultaneously in multiple terminals. Doing so results in failure to encrypt or decrypt all of the specified data. No data is lost, as the source data is not removed, but you must re-run the failed operations sequentially.

As with encryption, you can specify a directory instead of a file:

```
$ sudo /usr/sbin/navencrypt-move decrypt /path/to/encrypted/directory
```

Migrating eCryptfs-Encrypted Data to dm-crypt

As of August 2015, Filesystem-level encryption using eCryptfs is deprecated. Use this procedure to migrate to dm-crypt.

RHEL 7 does not support eCryptfs. For new installations on RHEL 7, you must use Block-Level Encryption with dm-crypt on page 315. If you are planning on upgrading to RHEL 7 and are currently using eCryptfs, migrate to dm-crypt before upgrading.

> **Warning:** Before encrypting or decrypting any data, stop all processes (for example, MySQL, MongoDB, PostgreSQL, and so on) that have access to the target data. **Failure to do so could lead to data corruption.**

1. Prepare an empty block device. This can be a physical block device (such as an unused disk) or a virtual block device (for example, a logical block device created by LVM, or a loop device). For instructions on creating a loop device, see Block-Level Encryption with a Loop Device on page 316.
2. Stop any services which depend on the encrypted data to be moved.
3. Prepare a block-level encrypted mount point. See Preparing for Encryption Using Cloudera Navigator Encrypt on page 313 for details about the procedure.
4. Add ACL rules for the new encrypted mount point that match the ACL rules for the mount point you are migrating from. To view existing ACL rules, run `sudo navencrypt acl --print`.

5. Add an ACL rule for your preferred shell (for example, `/bin/bash`) to enable command-line utilities such as `mv` and `cp`:

```
$ sudo navencrypt acl --add --rule="ALLOW @category * /bin/bash"
```

6. Copy the encrypted data from the eCryptfs mount point to the dm-crypt mount point:

```
$ sudo cp -rp /ecryptfs/mountpoint/path/to/data /dmcrypt/mountpoint/path/to/data
```

7. Update any symbolic links referencing the encrypted data. The following example demonstrates updating a symbolic link for a PostgreSQL database that was originally encrypted using eCryptfs, but has been migrated to dm-crypt:

```
$ sudo ls -l /var/lib/db/data/base/16385
lrwxrwxrwx 1 root root 72 Jul 22 15:33 /var/lib/db/data/base/16385 ->
/ecryptfs/mountpoint/postgres/var/lib/db/data/base/16385
$ sudo ln -sif /dmcrypt/mountpoint/postgres/var/lib/db/data/base/16385
/var/lib/db/data/base/16385
$ sudo ls -l /var/lib/db/data/base/16385
lrwxrwxrwx 1 root root 72 Jul 22 15:33 /var/lib/db/data/base/16385 ->
/dmcrypt/mountpoint/postgres/var/lib/db/data/base/16385
```

8. Remove the ACL rule enabling command-line utilities:

```
$ sudo navencrypt acl --del --rule="ALLOW @category * /bin/bash"
```

9. Restart any services which depend on the encrypted data.
10. Verify that the data was successfully copied, then delete the original eCryptfs-encrypted data. *Do not* delete any data until you are certain that you no longer need the original data.

   a. Stop the `navencrypt-mount` service:

```
$ sudo service navencrypt-mount stop
```

   b. Remove the original mountpoint directory and the storage directory with the original encrypted data.
   c. Edit `/etc/navencrypt/ztab` and remove entries for the original encrypted directory where `eCryptfs` is listed as the `<type>`.

   > **Important:** Use caution when editing the `/etc/navencrypt/ztab` file. Entries are tab-separated (not space-separated). The `ztab` file must not contain empty lines.

   d. Start the `navencrypt-mount` service:

```
$ sudo service navencrypt-mount start
```

Navigator Encrypt Access Control List

Managing the Access Control List

Cloudera Navigator Encrypt manages file system permissions with an access control list (ACL). This ACL is a security access control created by Cloudera that enables a predefined Linux process to access a file or directory managed by Navigator Encrypt.

The ACL uses rules to control process access to files. The rules specify whether a Linux process has access permissions to read from or write to a specific Navigator Encrypt path.

A rule is defined in the following order:

```
# TYPE @CATEGORY PATH PROCESS PARAMETERS
```

The following table defines the ACL rule components:

Table 29: ACL Rule Components

Component	Description
TYPE	Specifies whether to allow or deny a process. It can have either of the following values: `ALLOW` or `DENY`.
@CATEGORY	This is a user-defined shorthand, or container, for the encrypted dataset that the process will have access to. For example, if you are encrypting the directory `/var/lib/mysql`, you could use the category `@mysql` to indicate that this rule is granting access to a process on the MySQL data. See Listing Categories on page 330 for instructions on viewing existing categories.
PATH	Specifies the rights permissions of a specific path. For example: `*`, `www/*.htaccess`. Omit the leading slash (`/`).
PROCESS	Specifies the process or command name for the rule.
PARAMETERS	Tells the process the parent-child process to be executed: `--shell` defines the script for Navigator Encrypt to allow for executable process. Supported shells are `/usr/bin/bash`, `/bin/bash`, `/usr/bin/dash`, and `/bin/bash`. `--children` defines for Navigator Encrypt which child processes to allow that are executed by a process/script. Example: `--shell=/bin/bash`, `--children=/bin/df,/bin/ls`

All rules are stored in an encrypted policy file together with a set of process signatures that are used by Navigator Encrypt to authenticate each Linux process. This file is encrypted with the Navigator Encrypt key you defined during installation.

Cloudera recommends using `permissive` mode to assist with the initial ACL rule creation for your environment. In `permissive` mode, Navigator Encrypt allows full access to the encrypted data by all processes, but logs them in `dmesg` as `action="denied"` messages. Consult these messages to identify required ACL rules. To set Navigator Encrypt to permissive mode, use the following command:

```
$ sudo /usr/sbin/navencrypt set --mode=permissive
```

To view the current mode, run `navencrypt status -d`. For more information on access modes, see Access Modes.

deny2allow

After you generate the `action="denied"` messages, use the `navencrypt deny2allow` command to show which ACL rules are required, based on the `action="denied"` messages in `dmesg`. To show which ACL rules are required, perform the following steps:

1. Save the `dmesg` content to a file:

```
$ sudo dmesg > /tmp/dmesg.txt
```

2. Use the `dmesg.txt` file content as input to the `deny2allow` command to analyze the `action="denied"` messages and display a list of required ACL rules based on the `action="denied"` messages. Here is an example command and output:

```
$ sudo /usr/sbin/navencrypt deny2allow /tmp/dmesg.txt
ALLOW @mysql employees/* /usr/sbin/mysqld
ALLOW @mysql * /bin/bash
ALLOW @mysql * /bin/ls
```

If you need to clear the `dmesg` log and start fresh, run `dmesg -c`.

If a rule is displayed in the output from the command, it does not automatically mean the ACL rule must be added. You must determine which rules are actually needed. For example, the rule for `ls` would not typically be added as an ACL rule.

After the initial ACL rules are created, disable `permissive` mode with the following command:

```
$ sudo /usr/sbin/navencrypt set --mode=enforcing
```

Adding ACL Rules

Rules can be added one at a time using the command line or by specifying a policy file containing multiple rules. The following example shows how to add a single rule using the `navencrypt acl --add` command:

```
$ sudo /usr/sbin/navencrypt acl --add --rule="ALLOW @mysql * /usr/sbin/mysqld"
```

See Listing Categories on page 330 for instructions on viewing existing categories.

The following example shows how to add multiple rules using a policy file:

```
$ sudo /usr/sbin/navencrypt acl --add --file=/mnt/private/acl_rules
```

The contents of the policy file should contain one rule per line. For example:

```
ALLOW @mysql * /usr/sbin/mysqld
ALLOW @log * /usr/sbin/mysqld
ALLOW @apache * /usr/lib/apache2/mpm-prefork/apache2
```

Navigator Encrypt releases 3.10 and higher support comments in the policy file. Comments begin with the hash (#) symbol. You can use comments to annotate the policy file, or to temporarily disable a rule for testing. For example:

```
# Cloudera Navigator Encrypt policy file
# Allow mysqld to access all database files
ALLOW @mysql * /usr/sbin/mysqld
# Allow mysqld to write logs
ALLOW @log * /usr/sbin/mysqld
# ALLOW @apache * /usr/lib/apache2/mpm-prefork/apache2
```

Using a policy file is the fastest way to add multiple rules because it only requires the security key one time.

It is also possible to overwrite the entire current rules set with the option `--overwrite`. When this command is executed, all current rules are replaced by the ones specified in the file that contains the new set of rules. Cloudera recommends to save a copy of your current set of rules by printing it with the option `--print`.

Here is an example command using the `--overwrite` option:

```
$ sudo /usr/sbin/navencrypt acl --overwrite --file=/mnt/private/acl_rules
```

Adding ACL Rules by Profile

If your environment requires more granular controls on the processes that can access the data, you can add extra controls by using profiles. Profiles set requirements on a process other than just having the correct fingerprint. They can include such things as process owner and group, required open files, and the current working directory. To see more about adding rules by profile, see ACL Profile Rules on page 326.

Deleting ACL Rules

Rules can be deleted in one of two ways:

1. Manually specifying the rule to delete using the command line.
2. Specifying the line number of the rule to delete.

The following example shows how to delete a rule by passing it as a parameter:

```
$ sudo /usr/sbin/navencrypt acl --del --rule="ALLOW @mysql * /usr/sbin/mysqld "
```

If you remove a MySQL `ALLOW` rule, the MySQL cache must be cleaned by executing the `FLUSH TABLES;` MySQL statement. Otherwise, it will still be possible to view data from encrypted table.

The following example shows how to delete a rule by specifying a line number:

```
$ sudo /usr/sbin/navencrypt acl --del --line 3
```

It is also possible to delete multiple ACL rules in a single command:

```
$ sudo /usr/sbin/navencrypt acl --del --line=1,3
```

See Printing ACL Rules on page 325 for information on determining line numbers.

Deleting ACL Rules by Profile

See ACL Profile Rules on page 326 for instructions on deleting rules by profile.

Printing ACL Rules

You can print the current Access Control List using the following command:

```
$ sudo /usr/sbin/navencrypt acl --print
```

Save the ACL to a file with the `--file` option:

```
$ sudo /usr/sbin/navencrypt acl --print --file=policy-backup
```

To display additional information about the organization of the policy file, use the `--list` option:

```
$ sudo /usr/sbin/navencrypt acl --list
```

Universal ACL Rules

Universal ACLs will allow or deny a process access to all files or directories encrypted with Navigator Encrypt.

The rule `ALLOW @* * /process` allows the designated process to access anything from all encrypted categories.

The rule `ALLOW @data * *` allows all processes access to any path under the `@data` category.

The rule `ALLOW @* * *` allows all processes access to all encrypted categories. Cloudera does not recommend using this rule. Use it only in test environments.

Here is an example adding a universal ACL rule and then displaying it:

```
$ sudo /usr/sbin/navencrypt acl --add --rule="ALLOW @* * /usr/sbin/mysqld"
Type MASTER passphrase:
1 rule(s) were added
# navencrypt acl --listType MASTER passphrase:
# -  Type     Category       Path                        Profile  Process
1    ALLOW    @*             *                                    /usr/sbin/mysqld
```

Enabling Shell Scripts to Be Detected by ACL

All of the previous rules work for binary files. There may be times a script, such as a shell script, must be allowed to access the encrypted directory.

You can add the script as a rule by indicating the executable binary process of this script using the `--shell` option, for example:

```
ALLOW @scripts * /root/script.sh --shell=/bin/bash
```

The `--shell` option identifies which executable process is used to execute the script. Supported shells are `/usr/bin/bash`, `/bin/bash`, `/usr/bin/dash`, and `/bin/bash`

If the script is altered, it will no longer be trusted by the ACL because the fingerprint has changed. If you edit the script you must invoke the update option to update the ACL with the new fingerprint.

In some cases, it may be necessary to grant permissions to sub-processes invoked by scripts. For example, it may be necessary to grant permissions to `/bin/bash` that also allow running the `/bin/df` command to allow the system administrator to check disk capacity through a script run using a `crontab` entry. By using the `--children` option, you can specify these permissions. For example:

```
ALLOW @scripts * /root/script.sh --shell=/bin/bash --children=/bin/df
```

The `--children` option tells Navigator Encrypt to allow the `/bin/df` binary process if it is executed by `/root/script.sh`.

To allow more than one sub-process, identify them with the `--children` option as comma-separated values. For example:

```
ALLOW @scripts * /root/script.sh --shell=/bin/bash --children=/bin/df,/bin/ls
```

To add shell-children sub-processes, execute the `navencrypt acl --add` command, for example:

```
$ sudo /usr/sbin/navencrypt acl --add --rule="ALLOW @mysql * /usr/bin/mysqld_safe \
--shell=/bin/bash --children=/bin/df,/bin/ls"
```

ACL Profile Rules

If your environment requires more granular controls on the processes that can access the data, you can add extra controls by using profiles. Profiles set requirements on a process other than just having the correct fingerprint. They can include such things as process owner and group, required open files, and the current working directory.

A profile is generated by using the following command:

```
$ usr/sbin/navencrypt-profile --pid=<pid>
```

The output, by default, will be displayed on the screen. You can redirect the output to a file using the > or >> redirect operators. You can then edit the JSON output in the file to remove lines you do not want. By default, the profile includes the UID, the short name of the binary or script (identified as comm), and the full command line of the running process (including any parameters passed). You can generate information by using one of these flags:

- `-c`, `--with-cwd`

Output the current working directory

- `-e`, `--with-egid`

  Output the egid

- `-g`, `--with-gid`

  Output the gid

- `-u`, `--with-euid`

  Output the euid

Example output from the `navencrypt-profile` command:

```
{
"uid":"0",
"comm":"NetworkManager",
"cmdline":"NetworkManager –pid-file=/var/run/NetwrkManager/NetworkManager.pid",
"gid":"0"
"cwd":"/",
"fd0":"/dev/null",
"fd1":"/dev/null",
"fd2":"/dev/null"
}
```

Some distributions do not support `euid` and `guid`. Make sure that your profile file is correct by executing the following command to verify the expected IDs:

```
$ ps -p <pid_of_process> -o euid,egid
```

If `cmdline` parameters are variable, such as appending a process start timestamp to a filename, then the process profile will not match on subsequent restarts of the process because the current profile will have an updated timestamp and access will be denied by the ACL. You can mark those parameters as variable inside the profile file. For example, if the `cmdline` of a process is something like this:

```
"cmdline":"NetworkManager –pid-file=/var/run/NetworkManager/NetworkManager.pid \
-logfile=/var/log/NetworkManager/log-20130808152300.log"
```

Where `log-20130505122300.log` is a variable `cmdline` parameter, before adding the process profile to the ACL, edit the process profile file and use ## to specify that a particular parameter is variable:

```
"cmdline":"NetworkManager –pid-file=/var/run/NetworkManager/NetworkManager.pid
-logfile=##"
```

With the above configuration, the ACL will allow any value for the `-logfile` `cmdline` parameter.

To enable a profile in the ACL, use the additional parameter `--profile-file=<filename>` when adding the rule to the ACL:

```
$ sudo /usr/sbin/navencrypt acl --add --rule="ALLOW @mysql * /usr/sbin/mysqld" \
–-profile-file=/path/to/profile/file
```

To display the profile portion of the rules, use the `--all` parameter with `navencrypt acl --list`:

```
$ sudo /usr/sbin/navencrypt acl --list --all
Type MASTER passphrase:
# - Type Category Path Profile Process
1 ALLOW @mysql * YES /usr/sbin/mysqld
PROFILE:
{"uid":"120","comm":"mysqld","cmdline":"mysqld"}
```

Maintaining Cloudera Navigator Encrypt

Backing Up Navigator Encrypt

Perform backups of encrypted data, mount points, and the Navigator Encrypt configuration directory (`/etc/navencrypt`) on a regular basis.

> **Warning:** Failure to back up the configuration directory makes your backed-up encrypted data unrecoverable in the event of data loss.

Validating Navigator Encrypt Configuration

To validate the Navigator Encrypt deployment, run the following command:

```
$ sudo navencrypt status --integrity
```

This command verifies that:

- The mount encryption key (MEK) exists for each mount point.
- Each mount point in `/etc/navencrypt/ztab` has a corresponding entry in the control file (`/etc/navencrypt/control`).
- Each mount point directory exists.
- For loop devices, the file used for encrypted storage exists.

The output is similar to the following:

```
$ sudo navencrypt status --integrity
Checking MEKs integrity

        Mountpoint: /dev/loop0
                MEK file exist: ......................   [YES]
        Mountpoint: /dev/loop1
                MEK file exist: ......................   [YES]

Checking Ztab Mountpoints integrity

         Mountpoint: /dev/loop0
                 ztab vs control correspondence: .....   [YES]
                 Mountpoint directory exists: ........   [YES]
         Mountpoint: /dev/loop1
                 ztab vs control correspondence: .....   [YES]
                 Mountpoint directory exists: ........   [YES]

Checking Datastore backend files

         Datastore: '/root/my_storage_test'
                 Backend file exist: .................   [YES]
```

Restoring Mount Encryption Keys (MEKs) and Control File

Navigator Encrypt deposits its mount encryption keys (MEKs) and control file (`/etc/navencrypt/control`) in Cloudera Navigator Key Trustee Server. If these files are accidentally deleted, they can be restored from Key Trustee Server using the following commands:

- To restore MEKs:

```
$ sudo navencrypt key --restore-keys
```

- To restore the control file:

```
$ sudo navencrypt control --restore-control-file
```

Access Modes

Navigator Encrypt provides three different access modes:

- **Enforcing** is the default mode in which Navigator Encrypt validates access from all processes against the ACL. To protect your data, enforcing mode must be enabled.
- **Permissive** mode causes `action="denied"` messages to be logged in `dmesg`. It **does not** prevent access to the encrypted data. This mode is a dry-run feature to run and build ACL rules.
- **Admin** mode, as well as permissive mode, **does not** prevent access to the encrypted data. It allows any process to access the information because the ACL rules are not validated against the process. Admin mode does not cause `action="denied"` messages to be logged in `dmesg`.

To view the current access mode, run the following command:

```
$ sudo /usr/sbin/navencrypt status -d
```

> **Note:** The `navencrypt status` command reports that the `navencrypt` module is not running if no volumes are encrypted or the kernel module is not loaded.

To change the access mode, use the following command:

```
$ sudo /usr/sbin/navencrypt set --mode={enforcing|permissive|admin}
```

You cannot change the Navigator Encrypt access mode unless the Navigator Encrypt module is running. To view the status of the Navigator Encrypt module, run `navencrypt status --module`.

To start the Navigator Encrypt module there must be at least one active mount-point. To verify the mount-points status, run the following command:

```
$ sudo /etc/init.d/navencrypt-mount status
```

For RHEL 7, use `systemctl` instead:

```
$ sudo systemctl status navencrypt-mount
```

Changing and Verifying the Master Key

You can perform two operations with the `navencrypt key` command: `change` and `verify`.

You can verify a key against the Navigator Encrypt module, the Navigator Key Trustee server, or both. For example:

```
$ sudo /usr/sbin/navencrypt key --verify
$ sudo /usr/sbin/navencrypt key --verify --only-module
$ sudo /usr/sbin/navencrypt key --verify --only-keytrustee
```

> **Note:** The `navencrypt key` command fails if no volumes are encrypted or the kernel module is not loaded.

The master key can be changed in the event that another key-type authentication mechanism or a new master key is required. Valid master key types are single-passphrase, dual-passphrase, and RSA key files. To change the master key type, issue the following command and follow the interactive console:

```
$ sudo /usr/sbin/navencrypt key –-change
```

You can use the `--trustees`, `--votes`, and `--recoverable` options for the new key as described in [Table 26: Registration Options](#) on page 310.

Listing Categories

To list the existing categories for each mount point, run the command `navencrypt-move --list-categories`. For example:

```
$ sudo navencrypt-move --list-categories
Navigator Encrypt Categories found per Mountpoint:

  /dmcrypt-storage
     @mysql
     @keytabs

  /home/jdoe/secrets
     @moms_recipes
     @world_domination_plan
```

Updating ACL Fingerprints

All rules reference a process fingerprint (a SHA256 digest) that is used to authenticate the process into the file system. If the filesystem detects a fingerprint that is different from the one stored in the ACL, the Linux process is denied access and treated as an untrusted process.

Occasionally this process fingerprint must be updated, such as when software is upgraded. When the fingerprint must be updated, the Navigator Encrypt administrator re-authenticates the process on the ACL by executing the command `navencrypt acl --update`.

The following example demonstrates how to determine when a process fingerprint has been changed and must be updated:

```
$ sudo /usr/sbin/navencrypt acl --list
Type MASTER passphrase:
# -  Type Category    Path         Profile  Process
1 !! ALLOW @mysql *            /usr/sbin/mysqld
2 ALLOW @log * /usr/sbin/mysqld
3 !! ALLOW @apache * /usr/lib/apache2/mpm-prefork/
```

In the example above, the double exclamation (`!!`) characters indicate that a process fingerprint has changed and must be updated. Similarly, double E (EE) characters indicate a process read error. This error can be caused by a process that does not exist or that has permission issues.

Note:

For RHEL-compatible OSes, the `prelink` application may also be responsible for ACL fingerprint issues. Prelinking is intended to speed up a system by reducing the time a program needs to begin. Cloudera highly recommends disabling any automated `prelink` jobs, which are enabled by default in some systems. As an alternative, Cloudera recommends that you integrate a manual `prelink` run into your existing change control policies to ensure minimal downtime for applications accessing encrypted data.

To disable prelinking, modify the `/etc/sysconfig/prelink` file and change `PRELINKING=yes` to `PRELINKING=no`. Then, run the `/etc/cron.daily/prelink` script as root. Once finished, automatic prelinking is disabled.

For more information about how prelinking affects your system, see prelink.

Managing Mount Points

The `/etc/init.d/navencrypt-mount` command mounts all mount points that were registered with the `navencrypt-prepare` command and are listed in the `/etc/navencrypt/ztab` file. The possible operations are:

- `start`
- `stop`
- `status`
- `restart`

For RHEL 7, use `systemctl [start|stop|status|restart] navencrypt-mount`.

Note: Do not manually unmount the encryption mount point (for example, using `umount`). If you do so, you must manually close the `dm-crypt` device using the following procedure:

1. Run `dmsetup table` to list the `dm-crypt` devices.
2. Run `cryptsetup luksClose` *<device_name>* to close the device for the unmounted mount point.

When executing the `stop` operation, the encrypted mount point is unmounted, and your data becomes inaccessible.

The following example shows how to execute `navencrypt-mount status` with some inactive mount points:

```
$ sudo /etc/init.d/navencrypt-mount status
```

The following example shows how to execute the `navencrypt-mount stop` command:

```
$ sudo /etc/init.d/navencrypt-mount stop
```

The following example shows how to execute the `navencrypt-mount start` command:

```
$ sudo /etc/init.d/navencrypt-mount start
```

Here is an example command used to manually mount a directory:

```
$ sudo /usr/sbin/mount.navencrypt /path/to_encrypted_data/ /path/to/mountpoint
```

This command can be executed only if the `navencrypt-prepare` command was previously executed.

Allowing Access to a Single Operation

Sometimes running a single operation on encrypted data is required, such as listing the encrypted files or copying a particular file. The `navencrypt exec` command provides an efficient method to accomplish this. In the following command example, permission is denied when the command is run without `navencrypt exec`:

```
$ ls /mnt/db_encrypted/mysql/
ls: cannot open directory /mnt/db_encrypted/mysql/: Permission denied
```

If the `ls` command is executed with `navencrypt exec`, access is allowed one time only:

```
$ sudo /usr/sbin/navencrypt exec "ls -l /mnt/db_encrypted/mysql/"
```

Navigator Encrypt Kernel Module Setup

If the kernel headers were not installed on your host, or if the wrong version of the kernel headers were installed, the Navigator Encrypt module was not built at installation time. To avoid reinstalling the system, install the correct headers and execute the `navencrypt-module-setup` command. This attempts to build the module and install it.

This method is also an efficient way to install any new Navigator Encrypt module feature or fix without otherwise modifying your current Navigator Encrypt environment.

Navigator Encrypt Configuration Directory Structure

The file and directory structure of `/etc/navencrypt` is as follows:

```
$ tree /etc/navencrypt/
/etc/navencrypt/
    control -> /etc/navencrypt/jSpi9SM65xUIIhrau1Nn8ZXmQhrrQ9e363EUz8HKiRs
    jSpi9SM65xUIIhrau1Nn8ZXmQhrrQ9e363EUz8HKiRs
    rules
    ztab
locust
    keytrustee
     clientname
     deposits
         dev.loop0
        media.31E5-79B9locustlocust[system ~]# . /etc/*release[system ~]# . /etc/*release

         mnt.a
         mnt.encrypted
         mnt.tomount
     pubring.gpg
     pubring.gpg~
     random_seed
     secring.gpg
     trustdb.gpg
     ztrustee.conf
```

The following files and folders are part of the created file structure:

- `control`

  File that saves information about the mount points and corresponding Navigator Key Trustee keys. If this file is accidentally deleted, you can restore it using the `navencrypt control --restore-control-file` command.

- `rules`

  File that contains the ACL rules. It is encrypted with the user-provided master key.

- `ztab`

  File that contains information about all the mount points and their encryption type.

> **Important:** Use caution when editing the `/etc/navencrypt/ztab` file. Entries are tab-separated (not space-separated). The `ztab` file must not contain empty lines.

- `keytrustee`

  Directory where Navigator Key Trustee GPG keys are stored. These are generated during `navencrypt register` operations.

- `keytrustee/deposits`

  Directory where the Navigator Encrypt mount encryption keys (MEKs) are saved. These are encrypted with the user-provided master key. If these are accidentally deleted, you can can restore them from Key Trustee Server using the `navencrypt key --restore-keys` command.

Every mount point has an internal randomly-generated encryption passphrase.

Collecting Navigator Encrypt Environment Information

When troubleshooting problems with Navigator Encrypt, it is helpful to gather information about the installation and environment. Navigator Encrypt provides a command to facilitate this:

```
$ sudo navencrypt-collect
```

This command collects and outputs to the console the following information:

- Information about the system on which Navigator Encrypt is installed
- Entries from `/etc/navencrypt/ztab`
- The contents of the `keytrustee.conf` file
- Recent entries from the Navigator Encrypt log file
- Configured software repositories
- Checksums of all `/usr/src/navencrypt*` and `/usr/sbin/navencrypt*` files

You can use this information to compare Navigator Encrypt installations and to provide to Cloudera Support for troubleshooting. The `navencrypt-collect` command only outputs this information on the console, and does not generate any files or upload to Cloudera.

To save the information to a file, use the redirect operator (>). For example:

```
$ sudo navencrypt-collect > navencrypt.info
```

Upgrading Navigator Encrypt Hosts

See Best Practices for Upgrading Navigator Encrypt Hosts for considerations when upgrading operating systems (OS) and kernels on hosts that have Navigator Encrypt installed.

Configuring Encryption for Data Spills

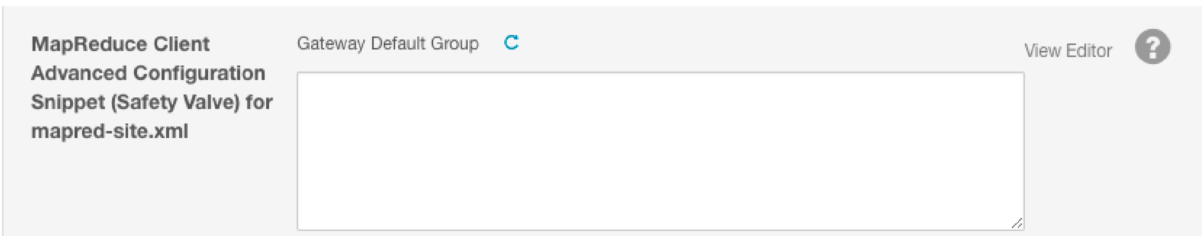
Some CDH services can encrypt data stored temporarily on the local filesystem outside HDFS. For example, data may *spill* to disk during memory-intensive operations, or when a service exceeds its allotted memory on a host. You can enable on-disk spill encryption for the following services.

MapReduce v2 (YARN)

MapReduce v2 can encrypt intermediate files generated during encrypted shuffle and data spilled to disk during the map and reduce stages. To enable encryption on these intermediate files, use Cloudera Manager to specify settings for the **MapReduce Client Advanced Configuration Snippet (Safety Valve)** for the `mapred-site.xml` associated with a gateway node.

From the Cloudera Manager Admin Console:

1. Select **Clusters** > **YARN**.
2. Click the **Configuration** tab.
3. In the Search field, enter **MapReduce Client Advanced Configuration Snippet (Safety Valve)** to find the safety valve on one of YARN's gateway nodes:



4. Enter the XML in the field (or switch to Editor mode and enter each property and its value in the fields provided). A complete example of the XML is shown here:

```
<property>
    <name>mapreduce.job.encrypted-intermediate-data</name>
    <value>true</value>
</property>
<property>
    <name>mapreduce.job.encrypted-intermediate-data-key-size-bits</name>
    <value>128</value>
</property>
<property>
    <name>mapreduce.job.encrypted-intermediate-data.buffer.kb</name>
    <value>128</value>
</property>
```

5. Click **Save Changes**.

The table provides descriptions of the properties used for data-spill encryption:

Property	Default	Description
`mapreduce.job.encrypted-intermediate-data`	`false`	Enables (true) and disables (false) encryption for intermediate MapReduce spills.
`mapreduce.job.encrypted-intermediate-data-key-size-bits`	`128`	Length of key (bits) used for encryption.
`mapreduce.job.encrypted-intermediate-data.buffer.kb`	`128`	Buffer size (Kb) for the stream written to disk after encryption.

> **Note:** Enabling encryption for intermediate data spills limits the number of attempts for a job to 1.

HBase

HBase does not write data outside HDFS, and does not require spill encryption.

Impala

Impala allows certain memory-intensive operations to be able to write temporary data to disk in case these operations approach their memory limit on a host. For details, read SQL Operations that Spill to Disk. To enable disk spill encryption in Impala:

1. Go to the Cloudera Manager Admin Console.
2. Click the **Configuration** tab.
3. Select **Scope** > **Impala Daemon**.
4. Select **Category** > **Security**.
5. Check the checkbox for the **Disk Spill Encryption** property.
6. Click **Save Changes.**

Hive

Hive jobs occasionally write data temporarily to local directories. If you enable HDFS encryption, you must ensure that the following intermediate local directories are also protected:

- **`LOCALSCRATCHDIR`**: The MapJoin optimization in Hive writes HDFS tables to a local directory and then uploads them to the distributed cache. To ensure these files are encrypted, either disable MapJoin by setting `hive.auto.convert.join` to `false`, or encrypt the *local* Hive Scratch directory (`hive.exec.local.scratchdir`) using Cloudera Navigator Encrypt.
- **`DOWNLOADED_RESOURCES_DIR`**: JARs that are added to a user session and stored in HDFS are downloaded to `hive.downloaded.resources.dir` on the HiveServer2 local filesystem. To encrypt these JAR files, configure Cloudera Navigator Encrypt to encrypt the directory specified by `hive.downloaded.resources.dir`.
- **NodeManager Local Directory List:** Hive stores JARs and MapJoin files in the distributed cache. To use MapJoin or encrypt JARs and other resource files, the `yarn.nodemanager.local-dirs` YARN configuration property must be configured to a set of encrypted local directories on all nodes.

For more information on Hive behavior with HDFS encryption enabled, see Using HDFS Encryption with Hive.

Flume

Flume supports on-disk encryption for log files written by the Flume file channels. See Configuring Encrypted On-disk File Channels for Flume on page 335.

Configuring Encrypted On-disk File Channels for Flume

Flume supports on-disk encryption of data on the local disk. To implement this:

- Generate an encryption key to use for the Flume Encrypted File Channel
- Configure on-disk encryption by setting parameters in the `flume.conf` file

Configuring Encryption for Data Spills

> **Important:**
>
> Flume on-disk encryption operates with a maximum strength of 128-bit AES encryption unless the JCE unlimited encryption cryptography policy files are installed. Please see this Oracle document for information about enabling strong cryptography:
> http://www.oracle.com/technetwork/java/javase/downloads/jce-6-download-429243.html
>
> Consult your security organization for guidance on the acceptable strength of your encryption keys. Cloudera has tested with AES-128, AES-192, and AES-256.

Generating Encryption Keys

Use the `keytool` program included with the Oracle JDK to create the AES encryption keys for use with Flume.

The command to generate a 128-bit key that uses the same password as the key store password is:

```
keytool -genseckey -alias key-1 -keyalg AES -keysize 128 -validity 9000 \
-keystore test.keystore -storetype jceks \
-storepass keyStorePassword
```

The command to generate a 128-bit key that uses a different password from that used by the key store is:

```
keytool -genseckey -alias key-0 -keypass keyPassword -keyalg AES \
-keysize 128 -validity 9000 -keystore test.keystore \
-storetype jceks -storepass keyStorePassword
```

The key store and password files can be stored anywhere on the file system; both files should have `flume` as the owner and `0600` permissions.

Please note that `-keysize` controls the strength of the AES encryption key, in bits; 128, 192, and 256 are the allowed values.

Configuration

Flume on-disk encryption is enabled by setting parameters in the `/etc/flume-ng/conf/flume.conf` file.

Basic Configuration

The first example is a basic configuration with an alias called `key-0` that uses the same password as the key store:

```
agent.channels.ch-0.type = file
agent.channels.ch-0.capacity = 10000
agent.channels.ch-0.encryption.cipherProvider = AESCTRNOPADDING
agent.channels.ch-0.encryption.activeKey = key-0
agent.channels.ch-0.encryption.keyProvider = JCEKSFILE
agent.channels.ch-0.encryption.keyProvider.keyStoreFile = /path/to/my.keystore
agent.channels.ch-0.encryption.keyProvider.keyStorePasswordFile =
/path/to/my.keystore.password
agent.channels.ch-0.encryption.keyProvider.keys = key-0
```

In the next example, `key-0` uses its own password which may be different from the key store password:

```
agent.channels.ch-0.type = file
agent.channels.ch-0.capacity = 10000
agent.channels.ch-0.encryption.cipherProvider = AESCTRNOPADDING
agent.channels.ch-0.encryption.activeKey = key-0
agent.channels.ch-0.encryption.keyProvider = JCEKSFILE
agent.channels.ch-0.encryption.keyProvider.keyStoreFile = /path/to/my.keystore
agent.channels.ch-0.encryption.keyProvider.keyStorePasswordFile =
/path/to/my.keystore.password
agent.channels.ch-0.encryption.keyProvider.keys = key-0
agent.channels.ch-0.encryption.keyProvider.keys.key-0.passwordFile =
/path/to/key-0.password
```

Changing Encryption Keys Over Time

To modify the key, modify the configuration as shown below. This example shows how to change the configuration to use `key-1` instead of `key-0`:

```
agent.channels.ch-0.type = file
agent.channels.ch-0.capacity = 10000
agent.channels.ch-0.encryption.cipherProvider = AESCTRNOPADDING
agent.channels.ch-0.encryption.activeKey = key-1
agent.channels.ch-0.encryption.keyProvider = JCEKSFILE
agent.channels.ch-0.encryption.keyProvider.keyStoreFile = /path/to/my.keystore
agent.channels.ch-0.encryption.keyProvider.keyStorePasswordFile =
/path/to/my.keystore.password
agent.channels.ch-0.encryption.keyProvider.keys = key-0 key-1
```

The same scenario except that `key-0` and `key-1` have their own passwords is shown here:

```
agent.channels.ch-0.type = file
agent.channels.ch-0.capacity = 10000
agent.channels.ch-0.encryption.cipherProvider = AESCTRNOPADDING
agent.channels.ch-0.encryption.activeKey = key-1
agent.channels.ch-0.encryption.keyProvider = JCEKSFILE
agent.channels.ch-0.encryption.keyProvider.keyStoreFile = /path/to/my.keystore
agent.channels.ch-0.encryption.keyProvider.keyStorePasswordFile =
/path/to/my.keystore.password
agent.channels.ch-0.encryption.keyProvider.keys = key-0 key-1
agent.channels.ch-0.encryption.keyProvider.keys.key-0.passwordFile =
/path/to/key-0.password
agent.channels.ch-0.encryption.keyProvider.keys.key-1.passwordFile =
/path/to/key-1.password
```

Troubleshooting

If the unlimited strength JCE policy files are not installed, an error similar to the following is printed in the `flume.log`:

```
07 Sep 2012 23:22:42,232 ERROR [lifecycleSupervisor-1-0]
(org.apache.flume.channel.file.encryption.AESCTRNoPaddingProvider.getCipher:137) - Unable
 to load key using transformation: AES/CTR/NoPadding; Warning: Maximum allowed key length
 = 128 with the available JCE security policy files. Have you installed the JCE unlimited
 strength jurisdiction policy files?
java.security.InvalidKeyException: Illegal key size
at javax.crypto.Cipher.a(DashoA13*..)
at javax.crypto.Cipher.a(DashoA13*..)
at javax.crypto.Cipher.a(DashoA13*..)
at javax.crypto.Cipher.init(DashoA13*..)
at javax.crypto.Cipher.init(DashoA13*..)
at
org.apache.flume.channel.file.encryption.AESCTRNoPaddingProvider.getCipher(AESCTRNoPaddingProvider.java:120)
at
org.apache.flume.channel.file.encryption.AESCTRNoPaddingProvider.access$200(AESCTRNoPaddingProvider.java:35)
at
org.apache.flume.channel.file.encryption.AESCTRNoPaddingProvider$AESCTRNoPaddingDecryptor.<init>(AESCTRNoPaddingProvider.java:94)
at
org.apache.flume.channel.file.encryption.AESCTRNoPaddingProvider$AESCTRNoPaddingDecryptor.<init>(AESCTRNoPaddingProvider.java:91)
at
org.apache.flume.channel.file.encryption.AESCTRNoPaddingProvider$DecryptorBuilder.build(AESCTRNoPaddingProvider.java:66)
at
org.apache.flume.channel.file.encryption.AESCTRNoPaddingProvider$DecryptorBuilder.build(AESCTRNoPaddingProvider.java:62)
at
org.apache.flume.channel.file.encryption.CipherProviderFactory.getDecrypter(CipherProviderFactory.java:47)
at org.apache.flume.channel.file.LogFileV3$SequentialReader.<init>(LogFileV3.java:257)
at
org.apache.flume.channel.file.LogFileFactory.getSequentialReader(LogFileFactory.java:110)
at org.apache.flume.channel.file.ReplayHandler.replayLog(ReplayHandler.java:258)
at org.apache.flume.channel.file.Log.replay(Log.java:339)
at org.apache.flume.channel.file.FileChannel.start(FileChannel.java:260)
at
org.apache.flume.lifecycle.LifecycleSupervisor$MonitorRunnable.run(LifecycleSupervisor.java:236)
at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:441)
at java.util.concurrent.FutureTask$Sync.innerRunAndReset(FutureTask.java:317)
```

```
at java.util.concurrent.FutureTask.runAndReset(FutureTask.java:150)
at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$101(ScheduledThreadPoolExecutor.java:98)
at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.runPeriodic(ScheduledThreadPoolExecutor.java:180)
at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:204)
at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
at java.lang.Thread.run(Thread.java:662)
```

Authorization

Authorization is concerned with who or what has access or control over a given resource or service. Since Hadoop merges together the capabilities of multiple varied, and previously separate IT systems as an enterprise data hub that stores and works on all data within an organization, it requires multiple authorization controls with varying granularities. In such cases, Hadoop management tools simplify setup and maintenance by:

- Tying all users to groups, which can be specified in existing LDAP or AD directories.
- Providing role-based access control for similar interaction methods, like batch and interactive SQL queries. For example, Apache Sentry permissions apply to Hive (HiveServer2) and Impala.

CDH currently provides the following forms of access control:

- Traditional POSIX-style permissions for directories and files, where each directory and file is assigned a single owner and group. Each assignment has a basic set of permissions available; file permissions are simply read, write, and execute, and directories have an additional permission to determine access to child directories.
- Extended Access Control Lists (ACLs) for HDFS that provide fine-grained control of permissions for HDFS files by allowing you to set different permissions for specific named users or named groups.
- Apache HBase uses ACLs to authorize various operations (`READ, WRITE, CREATE, ADMIN`) by column, column family, and column family qualifier. HBase ACLs are granted and revoked to both users and groups.
- Role-based access control with Apache Sentry. As of Cloudera Manager 5.1.x, Sentry permissions can be configured using either policy files or the database-backed Sentry service.
  - The Sentry service is the preferred way to set up Sentry permissions. See The Sentry Service on page 351 for more information.
  - For the policy file approach to configuring Sentry, see Sentry Policy File Authorization on page 392.

> **Important:** Cloudera does not support Apache Ranger or Hive's native authorization frameworks for configuring access control in Hive. Use Cloudera-supported Apache Sentry instead.

Cloudera Manager User Roles

Access to Cloudera Manager features is controlled by user accounts that specify an authentication mechanism and one or more user roles. User roles determine the Cloudera Manager capabilities that an authenticated user can perform and the features visible to the user in the Cloudera Manager Admin Console. Documentation for Cloudera Manager administration and management tasks indicate user roles required to perform the task.

> **Note:** All possible user roles are available with Cloudera Enterprise. Cloudera Express provides Read-Only and Full Administrator user roles only. When a Cloudera Enterprise Data Hub Edition trial license expires, only users with Read-Only and Full Administrator roles can log in to Cloudera Manager. A Full Administrator must change user accounts with other roles to Read-Only or Full Administrator before such users can log in.

Displaying Roles for Current User Account Login

The user roles associated with a given login session are available at any time from the Cloudera Manager Admin Console menu. Assuming you are logged in to Cloudera Manager Admin Console, you can always verify the user roles associated with your current login as follows:

1. Select **My Profile** from the ***username*** drop-down menu, where *username* is the name of the logged in account (such as `admin`). The **My Profile** pop-up window displays the Username, Roles, and the date and time of the Last Successful Login.

2. Click **Close** to dismiss the message page.

User Roles

A Cloudera Manager user account can be assigned one of the following roles with associated permissions:

- **Auditor**
  - View data in Cloudera Manager.
  - View audit events.
- **Read-Only**
  - View data in Cloudera Manager.
  - View service and monitoring information.

  The Read-Only role does not allow the user to add services or take any actions that affect the state of the cluster.
- **Dashboard**
  - Create, edit, or remove dashboards that belong to the user.
  - Add an existing chart or create a new chart to add to a dashboard that belongs to the user.

  Additionally, the Dashboard role can perform the same tasks as the Read-Only role.
- **Limited Operator**
  - View data in Cloudera Manager.
  - View service and monitoring information.
  - Decommission hosts (except hosts running Cloudera Management Service roles).

  The Limited Operator role does not allow the user to add services or take any other actions that affect the state of the cluster.
- **Operator**
  - View data in Cloudera Manager.
  - View service and monitoring information.
  - Stop, start, and restart clusters, services (except the Cloudera Management Service), and roles.
  - Decommission and recommission hosts (except hosts running Cloudera Management Service roles).
  - Decommission and recommission roles (except Cloudera Management Service roles).
  - Start, stop, and restart KMS.

  The Operator role does not allow the user to add services, roles, or hosts, or take any other actions that affect the state of the cluster.
- **Configurator**
  - View data in Cloudera Manager.
  - Perform all Operator operations.
  - Configure services (except the Cloudera Management Service).
  - Enter and exit maintenance mode.
  - Manage dashboards (including Cloudera Management Service dashboards).
  - Start, stop, and restart KMS
- **Cluster Administrator** - View all data and perform all actions *except* the following:
  - Administer Cloudera Navigator.
  - Perform replication and snapshot operations.
  - View audit events.
  - Manage user accounts and configuration of external authentication.
  - Manage Full Administrator accounts.

  - Configure HDFS encryption, administer Key Trustee Server, and manage encryption keys.
- **BDR Administrator**
  - View data in Cloudera Manager.
  - View service and monitoring information.
  - Perform replication and snapshot operations.
- **Navigator Administrator**
  - View data in Cloudera Manager.
  - View service and monitoring information.
  - Administer Cloudera Navigator.
  - View audit events.
- **User Administrator**
  - View data in Cloudera Manager.
  - View service and monitoring information.
  - Manage user accounts and configuration of external authentication.
- **Key Administrator**
  - View data in Cloudera Manager.
  - Configure HDFS encryption, administer Key Trustee Server, and manage encryption keys.
  - Start, stop, and restart KMS
- **Full Administrator** - Full Administrators have permissions to view all data and do all actions, including reconfiguring and restarting services, and administering other users.

Removing the Full Administrator User Role

Minimum Required Role: **User Administrator** (also provided by **Full Administrator**)

In some organizations, security policies may prohibit the use of the Full Administrator role. The Full Administrator role is created during Cloudera Manager installation, but you can remove it as long as you have at least one remaining user account with User Administrator privileges.

To remove the Full Administrator user role, perform the following steps.

1. Add at least one user account with User Administrator privileges, or ensure that at least one such user account already exists.
2. Ensure that there is only a single user account with Full Administrator privileges.
3. While logged in as the single remaining Full Administrator user, select your own user account and either delete it or assign it a new user role.

> **Warning:** After you delete the last Full Administrator account, you will be logged out immediately and will not be able to log in unless you have access to another user account. Also, it will no longer be possible to create or assign Full Administrators.

A consequence of removing the Full Administrator role is that some tasks may require collaboration between two or more users with different user roles. For example:

- If the machine that the Cloudera Navigator roles are running on needs to be replaced, the Cluster Administrator will want to move all the roles running on that machine to a different machine. The Cluster Administrator can move any non-Navigator roles by deleting and re-adding them, but would need a Navigator Administrator to perform the stop, delete, add, and start actions for the Cloudera Navigator roles.
- In order to take HDFS snapshots, snapshots must be enabled on the cluster by a Cluster Administrator, but the snapshots themselves must be taken by a BDR Administrator.

HDFS Extended ACLs

HDFS supports POSIX Access Control Lists (ACLs), as well as the traditional POSIX permissions model already supported. ACLs control access of HDFS files by providing a way to set different permissions for specific named users or named groups. They enhance the traditional permissions model by allowing users to define access control for arbitrary combinations of users and groups instead of a single owner/user or a single group.

Enabling HDFS Access Control Lists

By default, HDFS access control lists (ACLs) are disabled on a cluster. You can enable them using either Cloudera Manager or the command line.

> **Important:** Ensure that all users and groups resolve on the NameNode for ACLs to work as expected.

Enabling HDFS ACLs Using Cloudera Manager

1. Go to the Cloudera Manager Admin Console and navigate to the **HDFS** service.
2. Click the **Configuration** tab.
3. Select **Scope** > ***Service_name*** **(Service-Wide)**
4. Select **Category** > **Security**
5. Locate the **Enable Access Control Lists** property and select its checkbox to enable HDFS ACLs.
6. Click **Save Changes** to commit the changes.

Enabling HDFS ACLs Using the Command Line

To enable ACLs using the command line, set the `dfs.namenode.acls.enabled` property to `true` in the NameNode's `hdfs-site.xml`.

```
<property>
<name>dfs.namenode.acls.enabled</name>
<value>true</value>
</property>
```

Commands

To set and get file access control lists (ACLs), use the file system shell commands, `setfacl` and `getfacl`.

getfacl

```
hdfs dfs -getfacl [-R] <path>

<!-- COMMAND OPTIONS
<path>: Path to the file or directory for which ACLs should be listed.
-R: Use this option to recursively list ACLs for all files and directories.
-->
```

Examples:

```
<!-- To list all ACLs for the file located at /user/hdfs/file -->
hdfs dfs -getfacl /user/hdfs/file

<!-- To recursively list ACLs for /user/hdfs/file -->
hdfs dfs -getfacl -R /user/hdfs/file
```

setfacl

```
hdfs dfs -setfacl [-R] [-b|-k -m|-x <acl_spec> <path>]|[--set <acl_spec> <path>]

<!-- COMMAND OPTIONS
<path>: Path to the file or directory for which ACLs should be set.
-R: Use this option to recursively list ACLs for all files and directories.
-b: Revoke all permissions except the base ACLs for user, groups and others.
-k: Remove the default ACL.
-m: Add new permissions to the ACL with this option. Does not affect existing permissions.
-x: Remove only the ACL specified.
<acl_spec>: Comma-separated list of ACL permissions.
--set: Use this option to completely replace the existing ACL for the path specified.
       Previous ACL entries will no longer apply.
-->
```

Examples:

```
<!-- To give user ben read & write permission over /user/hdfs/file -->
hdfs dfs -setfacl -m user:ben:rw- /user/hdfs/file

<!-- To remove user alice's ACL entry for /user/hdfs/file -->
hdfs dfs -setfacl -x user:alice /user/hdfs/file

<!-- To give user hadoop read & write access, and group or others read-only access -->
hdfs dfs -setfacl --set user::rw-,user:hadoop:rw-,group::r--,other::r-- /user/hdfs/file
```

For more information on using HDFS ACLs, see the HDFS Permissions Guide on the Apache website.

Configuring LDAP Group Mappings

Each host that comprises a node in a Cloudera cluster runs an operating system, such as CentOS, Oracle Linux, and so on. At the OS-level are `user:group` accounts created during installation that map to the services running on that specific node of the cluster. The default shell-based group mapping provider, `org.apache.hadoop.security.ShellBasedUnixGroupsMapping`, handles the mapping from the local host system (the OS) to the specific cluster service, such as HDFS. The hosts authenticate using these local OS accounts before processes are allowed to run on the node.

For clusters integrated with Kerberos for authentication, the hosts must also provide Kerberos tickets before processes can run on the node. The cluster can use the organization's LDAP directory service to provide the login credentials, including Kerberos tickets, to authenticate transparently while the system runs. That means that the local `user:group` accounts on each host need to be mapped to LDAP accounts. To map local `user:group` accounts to an LDAP service:

- Use tools such as SSSD (Systems Security Services Daemon) or Centrify Server Suite (see Identity and Access management for Cloudera) rather than the Hadoop `LdapGroupsMapping`. Cloudera does not recommend using `LdapGroupsMapping` to replicate LDAP groups.
- Do not use Winbind to map Linux `user:group` accounts to Active Directory. It cannot scale, impedes cluster performance, and is not supported.
- Use the same `user:group` mappings across all cluster nodes, for ease of management.
- Use either Cloudera Manager or the command-line process detailed below.

The local user:group accounts must be mapped LDAP for group mappings in Hadoop, you must create the users and groups for your Hadoop services in LDAP.

To integrate the cluster with an LDAP service, the user:group relationships need to be contained in the LDAP directory, meaning, the admin must create the user accounts and define groups for user:group relationships on each host.

The user and group names listed in the table are the default user:group values for CDH services.

> **Note:** If the defaults have been changed for any service (by using the Cloudera Manager Admin Console to customize the **System User** or **System Group** setting for the service, for example), use the custom values to create the users and configure the group for that service in the LDAP server, rather than the defaults listed in the table below.

Cloudera Product or Component	User	Group
Cloudera Manager	`cloudera-scm`	`cloudera-scm`
Apache Accumulo	`accumulo`	`accumulo`
Apache Avro	(No default)	(No default)
Apache Flume (sink writing to HDFS needs write privileges)	`flume`	`flume`
Apache HBase (Master, RegionServer processes)	`hbase`	`hbase`
Apache HCatalog, WebHCat service	`hive`	`hive`
Apache Hive (HiveServer2, Hive Metastore)	`hive`	`hive`
Apache Kafka	`kafka`	`kafka`
Apache Mahout	(No default)	(No default)
Apache Oozie	`oozie`	`oozie`
Apache Pig	(No default)	(No default)
Apache Sentry	`sentry`	`sentry`
Apache Spark	`spark`	`spark`
Apache Sqoop1	`sqoop`	`sqoop`
Apache Sqoop2	`sqoop2`	`sqoop, sqoop2`
Apache Whirr	(No default)	(No default)
Apache ZooKeeper	`zookeeper`	`zookeeper`
Cloudera Impala	`impala`	`impala, hive`
Cloudera Search	`solr`	`solr`
HDFS (NameNode, DataNodes)	`hdfs`	`hdfs, hadoop`
HttpFS	`httpfs`	`httpfs`
Hue	`hue`	`hue`
Hue Load Balancer (needs apache2 package)	`apache`	`apache`
Java KeyStore KMS	`kms`	`kms`
Key Trustee KMS	`kms`	`kms`
Key Trustee Server	`keytrustee`	`keytrustee`
Kudu	`kudu`	`kudu`
Llama	`llama`	`llama`
MapReduce (JobTracker, TaskTracker)	`mapred`	`mapred, hadoop`
Parquet	(No default)	(No default)
YARN	`yarn`	`yarn, hadoop`

> **Note:** Cloudera Manager 5.3 (and later releases) can be setup for ***single user mode***. In single user mode, Hadoop users and groups are subsumed by `cloudera-scm:cloudera-scm`. Cloudera Manager starts all Cloudera Manager Agent processes and services running on the nodes in the cluster as a unit, owned by this `cloudera-scm:cloudera-scm`. Single user mode is not recommended for production clusters.

In addition:

- For Sentry with Hive, add these properties on the HiveServer2 node.
- For Sentry with Impala, add these properties to all hosts.

See Users and Groups in Sentry for more information.

Using Cloudera Manager

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

Make the following changes to the HDFS service's security configuration:

1. Open the Cloudera Manager Admin Console and go to the **HDFS** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HDFS (Service Wide)**
4. Select **Category** > **Security**.
5. Modify the following configuration properties using values from the table below:

Configuration Property	Value
Hadoop User Group Mapping Implementation	`org.apache.hadoop.security.LdapGroupsMapping`
Hadoop User Group Mapping LDAP URL	`ldap://<server>`
Hadoop User Group Mapping LDAP Bind User	`Administrator@example.com`
Hadoop User Group Mapping LDAP Bind User Password	`***`
Hadoop User Group Mapping Search Base	`dc=example,dc=com`

Although the above changes are sufficient to configure group mappings for Active Directory, some changes to the remaining default configurations might be required for OpenLDAP.

Using the Command Line

Add the following properties to the `core-site.xml` on the NameNode:

```
<property>
<name>hadoop.security.group.mapping</name>
<value>org.apache.hadoop.security.LdapGroupsMapping</value>
</property>

<property>
<name>hadoop.security.group.mapping.ldap.url</name>
<value>ldap://server</value>
</property>

<property>
<name>hadoop.security.group.mapping.ldap.bind.user</name>
<value>Administrator@example.com</value>
</property>

<property>
<name>hadoop.security.group.mapping.ldap.bind.password</name>
<value>****</value>
```

```
</property>

<property>
<name>hadoop.security.group.mapping.ldap.base</name>
<value>dc=example,dc=com</value>
</property>

<property>
<name>hadoop.security.group.mapping.ldap.search.filter.user</name>
<value>(&amp;(objectClass=user)(sAMAccountName={0}))</value>
</property>

<property>
<name>hadoop.security.group.mapping.ldap.search.filter.group</name>
<value>(objectClass=group)</value>
</property>

<property>
<name>hadoop.security.group.mapping.ldap.search.attr.member</name>
<value>member</value>
</property>

<property>
<name>hadoop.security.group.mapping.ldap.search.attr.group.name</name>
<value>cn</value>
</property>
```

Authorization With Apache Sentry

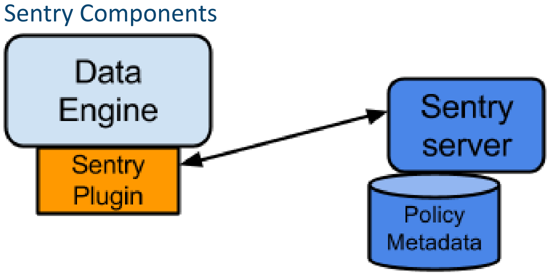
Apache Sentry is a granular, role-based authorization module for Hadoop. Sentry provides the ability to control and enforce precise levels of privileges on data for authenticated users and applications on a Hadoop cluster. Sentry currently works out of the box with Apache Hive, Hive Metastore/HCatalog, Apache Solr, Impala, and HDFS (limited to Hive table data).

Sentry is designed to be a pluggable authorization engine for Hadoop components. It allows you to define authorization rules to validate a user or application's access requests for Hadoop resources. Sentry is highly modular and can support authorization for a wide variety of data models in Hadoop.

Continue reading:

Architecture Overview

Sentry Components



There are three components involved in the authorization process:

- **Sentry Server**

  The Sentry RPC server manages the authorization metadata. It supports interfaces to securely retrieve and manipulate the metadata. In CDH 5.13 and above, you can configure multiple Sentry Servers for high availability. See Sentry High Availability on page 375 for information about how to enable high availability for Sentry.

- **Data Engine**

This is a data processing application, such as Hive or Impala, that needs to authorize access to data or metadata resources. The data engine loads the Sentry plugin and all client requests for accessing resources are intercepted and routed to the Sentry plugin for validation.

- **Sentry Plugin**

  The Sentry plugin runs in the data engine. It offers interfaces to manipulate authorization metadata stored in the Sentry Server, and includes the authorization policy engine that evaluates access requests using the authorization metadata retrieved from the server.

Key Concepts

- Authentication - Verifying credentials to reliably identify a user
- Authorization - Limiting the user's access to a given resource
- User - Individual identified by underlying authentication system
- Group - A set of users, maintained by the authentication system
- Privilege - An instruction or rule that allows access to an object
- Role - A set of privileges; a template to combine multiple access rules
- Authorization models - Defines the objects to be subject to authorization rules and the granularity of actions allowed. For example, in the SQL model, the objects can be databases or tables, and the actions are `SELECT`, `INSERT`, and `CREATE`. For the Search model, the objects are indexes, configs, collections, documents; the access modes include query and update.

User Identity and Group Mapping

Sentry relies on underlying authentication systems, such as Kerberos or LDAP, to identify the user. It also uses the group mapping mechanism configured in Hadoop to ensure that Sentry sees the same group mapping as other components of the Hadoop ecosystem.

Consider a sample organization with users Alice and Bob who belong to an Active Directory (AD) group called `finance-department`. Bob also belongs to a group called `finance-managers`. In Sentry, you first create roles and then grant privileges to these roles. For example, you can create a role called Analyst and grant `SELECT` on tables Customer and Sales to this role.

The next step is to join these authentication entities (users and groups) to authorization entities (roles). This can be done by granting the Analyst role to the `finance-department` group. Now Bob and Alice who are members of the `finance-department` group get `SELECT` privilege to the Customer and Sales tables.

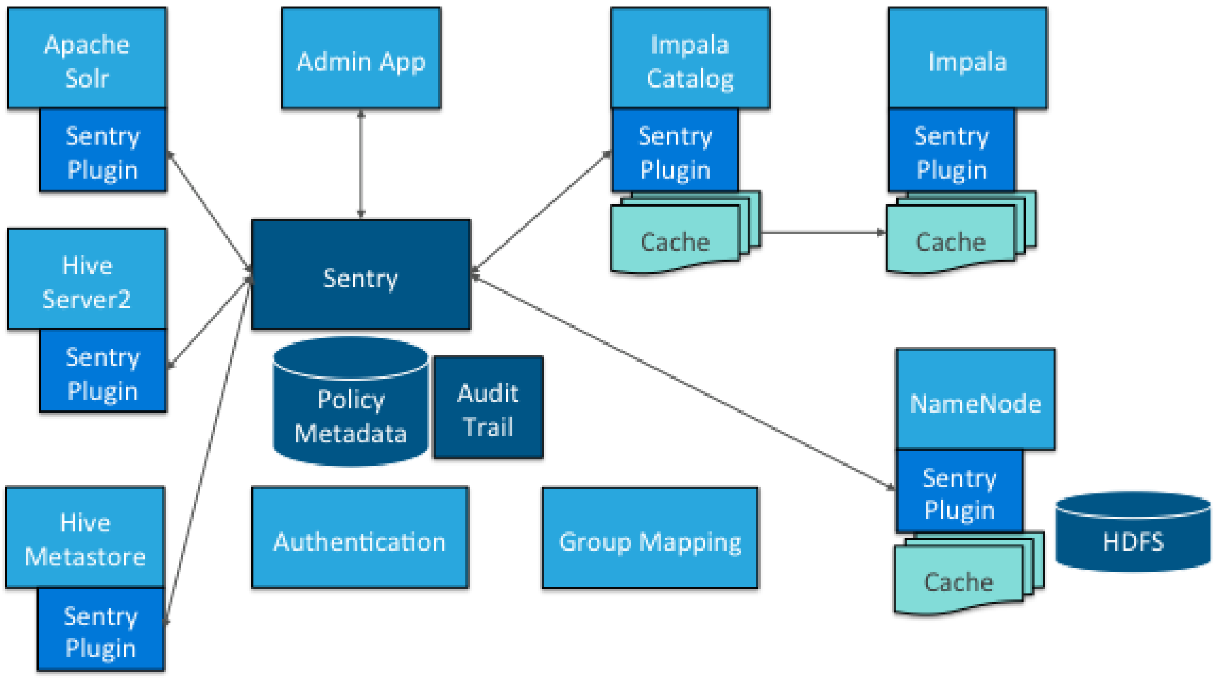
Role-Based Access Control

Role-based access control (RBAC) is a powerful mechanism to manage authorization for a large set of users and data objects in a typical enterprise. New data objects get added or removed, users join, move, or leave organisations all the time. RBAC makes managing this a lot easier. Hence, as an extension of the sample organization discussed previously, if a new employee Carol joins the Finance Department, all you need to do is add her to the `finance-department` group in AD. This will give Carol access to data from the Sales and Customer tables.

Unified Authorization

Another important aspect of Sentry is the unified authorization. The access control rules once defined, work across multiple data access tools. For example, being granted the Analyst role in the previous example will allow Bob, Alice, and others in the `finance-department` group to access table data from SQL engines such as Hive and Impala, as well as using MapReduce, Pig applications or metadata access using HCatalog.

Sentry Integration with the Hadoop Ecosystem



As illustrated above, Apache Sentry works with multiple Hadoop components. At the heart, you have the Sentry Server which stores authorization metadata and provides APIs for tools to retrieve and modify this metadata securely.

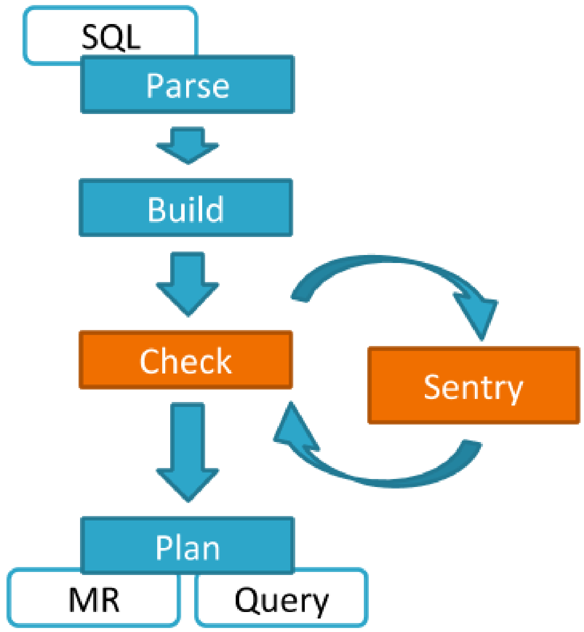
Note that the Sentry Server only facilitates the metadata. The actual authorization decision is made by a policy engine that runs in data processing applications such as Hive or Impala. Each component loads the Sentry plugin, which includes the service client for dealing with the Sentry service and the policy engine to validate the authorization request.

Hive and Sentry

Consider an example where Hive gets a request to access an object in a certain mode by a client. If Bob submits the following Hive query:

```
select * from production.sales
```

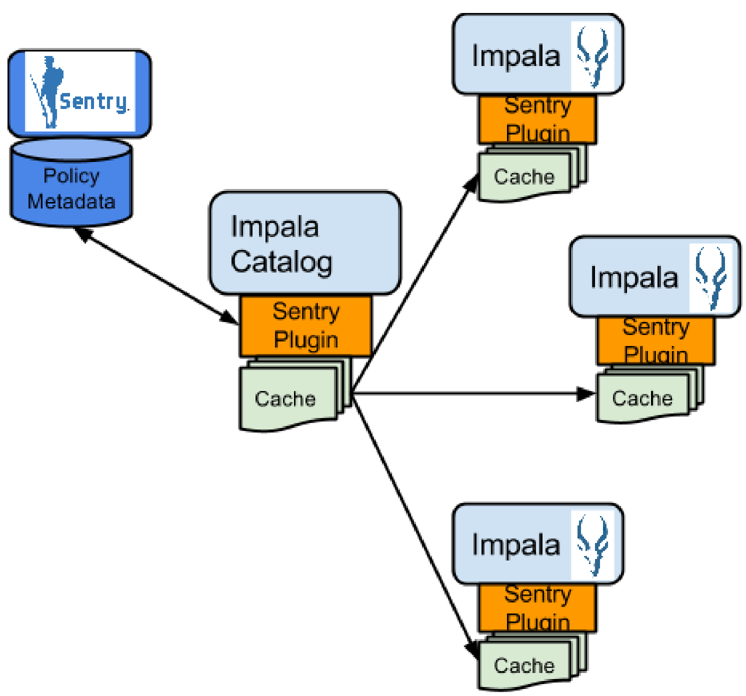
Hive will identify that user Bob is requesting `SELECT` access to the Sales table. At this point Hive will ask the Sentry plugin to validate Bob's access request. The plugin will retrieve Bob's privileges related to the Sales table and the policy engine will determine if the request is valid.



Hive works with both the Sentry service and policy files. Cloudera recommends you use the Sentry service, which makes it easier to manage user privileges. For more details and instructions, see The Sentry Service on page 351 *or* Sentry Policy File Authorization on page 392.

Impala and Sentry

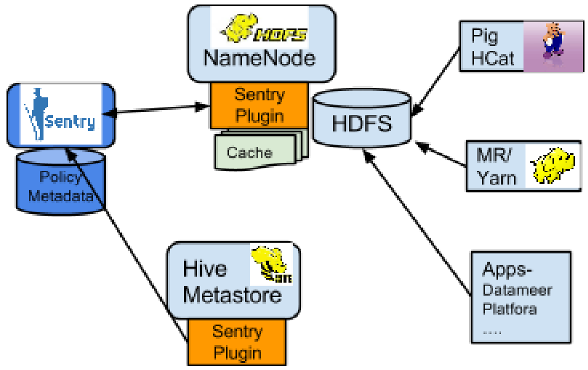
Authorization processing in Impala is similar to that in Hive. The main difference is caching of privileges. Impala's Catalog server manages caching schema metadata and propagating it to all Impala server nodes. This Catalog server caches Sentry metadata as well. As a result, authorization validation in Impala happens locally and much faster.



For detailed documentation, see Enabling Sentry Authorization for Impala on page 415.

Sentry-HDFS Synchronization

Sentry-HDFS authorization is focused on Hive warehouse data - that is, any data that is part of a table in Hive or Impala. The real objective of this integration is to expand the same authorization checks to Hive warehouse data being accessed from any other components such as Pig, MapReduce or Spark. At this point, this feature does not replace HDFS ACLs. Tables that are not associated with Sentry will retain their old ACLs.



The mapping of Sentry privileges to HDFS ACL permissions is as follows:

- SELECT privilege -> Read access on the file.
- INSERT privilege -> Write access on the file.
- ALL privilege -> Read and Write access on the file.

The NameNode loads a Sentry plugin that caches Sentry privileges as well Hive metadata. This helps HDFS to keep file permissions and Hive tables privileges in sync. The Sentry plugin periodically polls the Sentry and Metastore to keep the metadata changes in sync.

For example, if Bob runs a Pig job that is reading from the Sales table data files, Pig will try to get the file handle from HDFS. At that point the Sentry plugin on the NameNode will figure out that the file is part of Hive data and overlay Sentry privileges on top of the file ACLs. As a result, HDFS will enforce the same privileges for this Pig client that Hive would apply for a SQL query.

For HDFS-Sentry synchronization to work, you *must* use the Sentry service, not policy file authorization. See Synchronizing HDFS ACLs and Sentry Permissions on page 387, for more details.

Search and Sentry

Sentry can apply restrictions to various Search tasks including accessing data and creating collections. These restrictions are consistently applied, regardless of the way users attempt to complete actions. For example, restricting access to data in a collection restricts that access whether queries come from the command line, from a browser, or through the admin console.

With Search, Sentry restrictions can be stored in the database-backed Sentry service or in a policy file (for example, sentry-provider.ini) which is stored in an HDFS location such as `hdfs://ha-nn-uri/user/solr/sentry/sentry-provider.ini`.

Sentry with Search does not support multiple policy files for multiple databases. If you choose to use policy files rather than database-backed Sentry service, you must use a separate policy file for each Sentry-enabled service. For example, if Hive and Search were using policy file authorization, using a combined Hive and Search policy file would result in an invalid configuration and failed authorization on both services.

Search works with both the Sentry service and policy files. Cloudera recommends you use the Sentry service, which makes it easier to manage user privileges. For more details and instructions, see The Sentry Service on page 351 *or* Sentry Policy File Authorization on page 392.

For detailed documentation, see Configuring Sentry Authorization for Cloudera Search on page 425.

Authorization Administration

The Sentry Server supports APIs to securely manipulate roles and privileges. Both Hive and Impala support SQL statements to manage privileges natively. Sentry assumes that HiveServer2 and Impala run as superusers, usually called `hive` and `impala`. To initiate top-level permissions for Sentry, an admin must login as a superuser. You can use either Beeline or the Impala shell to execute the following sample statement:

```
GRANT ROLE Analyst TO GROUP finance-managers
```

Disabling Hive CLI

To execute Hive queries, you must use Beeline. Hive CLI is not supported with Sentry and therefore its access to the Hive Metastore must be disabled. This is especially necessary if the Hive metastore has sensitive metadata. To do this, modify the `hadoop.proxyuser.hive.groups` in `core-site.xml` on the Hive metastore host. For example, to give the `hive` user permission to impersonate only members of the `hive` and hue groups, set the property to:

```
<property>
<name>hadoop.proxyuser.hive.groups</name>
<value>hive,hue</value>
</property>
```

More user groups that require access to the Hive Metastore can be added to the comma-separated list as needed.

Using Hue to Manage Sentry Permissions

Hue supports a Security app to manage Sentry authorization. This allows users to explore and change table permissions. Here is a video blog that demonstrates its functionality.

The Sentry Service

The Sentry service is an RPC server that stores authorization metadata in an underlying relational database and provides RPC interfaces to retrieve and manipulate privileges. It supports secure access to services using Kerberos. The service serves authorization metadata from the database backed storage; it does not handle actual privilege validation. The Hive, Impala, and Solr services are clients of this service and will enforce Sentry privileges when they are configured to use Sentry.

This topic describes basic Sentry terminology and the Sentry privilege model:

For more information on installing, upgrading and configuring the Sentry service, see:

Terminology

- An *object* is an entity protected by Sentry's authorization rules. The objects supported in the current release are `server`, `database`, `table`, `URI`, `collection`, and `config`.
- A *role* is a collection of rules for accessing a given object.
- A *privilege* is granted to a role to govern access to an object. Sentry allows you to assign the `SELECT` privilege to columns (only for Hive and Impala). Supported privileges are:

Table 30: Valid privilege types and the objects they apply to

Privilege	Object
INSERT	DB, TABLE
SELECT	SERVER, DB, TABLE, COLUMN
UPDATE	COLLECTION, CONFIG
QUERY	COLLECTION, CONFIG
ALL	SERVER, TABLE, DB, URI, COLLECTION, CONFIG

- A *user* is an entity that is permitted by the authentication subsystem to access the service. This entity can be a Kerberos principal, an LDAP `userid`, or an artifact of some other supported pluggable authentication system.
- A *group* connects the authentication system with the authorization system. It is a collection of one or more users who have been granted one or more authorization roles. Sentry allows a set of roles to be configured for a group.
- A configured *group provider* determines a user's affiliation with a group. The current release supports HDFS-backed groups and locally configured groups.

Privilege Model

Sentry uses a role-based privilege model with the following characteristics.

- Allows any user to execute `show function`, `desc function`, and `show locks`.
- Allows the user to see only those tables, databases, collections, and configurations for which the user has privileges.
- Requires a user to have the necessary privileges on the URI to execute HiveQL operations that specify a location. Examples of such operations include `LOAD`, `IMPORT`, and `EXPORT`.
- Privileges granted on URIs are recursively applied to all subdirectories. That is, privileges only need to be granted on the parent directory.
- Sentry provides column-level access control for tables in Hive and Impala. You can assign the `SELECT` privilege on a subset of columns in a table.

Important:

- When Sentry is enabled, you must use Beeline to execute Hive queries. Hive CLI is not supported with Sentry and must be disabled.
- When Sentry is enabled, a user with no privileges on a database will not be allowed to connect to HiveServer2. This is because the `use <database>` command is executed as part of the connection to HiveServer2, which is why the connection fails. See HIVE-4256.

For more information, see Authorization Privilege Model for Hive and Impala.

User to Group Mapping

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

Group mappings in Sentry can be summarized as in the figure below.

The Sentry service only uses HadoopUserGroup mappings. See Configuring LDAP Group Mappings on page 343 for details on configuring LDAP group mappings in Hadoop.

Authorization Privilege Model for Hive and Impala

Privileges can be granted on different objects in the Hive warehouse. Any privilege that can be granted is associated with a level in the object hierarchy. If a privilege is granted on a container object in the hierarchy, the base object automatically inherits it. For instance, if a user has ALL privileges on the database scope, then (s)he has ALL privileges on all of the base objects contained within that scope. Sentry 5.13 and above supports Hive metastore high availability.

Object Hierarchy

```
Server
     URI
     Database
         Table
             Partition
             Columns
         View
```

Table 31: Valid privilege types and objects they apply to

Privilege	Object
INSERT	DB, TABLE
SELECT	DB, TABLE, VIEW, COLUMN
ALL	SERVER, TABLE, DB, URI

Table 32: Privilege hierarchy

Base Object	Granular privileges on object	Container object that contains the base object	Privileges on container object that implies privileges on the base object
DATABASE	ALL	SERVER	ALL
TABLE	INSERT	DATABASE	ALL
TABLE	SELECT	DATABASE	ALL
COLUMN	SELECT	DATABASE	ALL
VIEW	SELECT	DATABASE	ALL

Table 33: Privilege table for Hive & Impala operations

Operation	Scope	Privileges Required	URI
CREATE DATABASE	SERVER	ALL	
DROP DATABASE	DATABASE	ALL	
CREATE TABLE	DATABASE	ALL	
DROP TABLE	TABLE	ALL	
CREATE VIEW -This operation is allowed if you have column-level SELECT access to the columns being used.	DATABASE; SELECT on TABLE;	ALL	
ALTER VIEW -This operation is allowed if you have column-level SELECT access to the columns being used.	VIEW/TABLE	ALL	
DROP VIEW	VIEW/TABLE	ALL	
ALTER TABLE .. ADD COLUMNS	TABLE	ALL	
ALTER TABLE .. REPLACE COLUMNS	TABLE	ALL	
ALTER TABLE .. CHANGE column	TABLE	ALL	
ALTER TABLE .. RENAME	TABLE	ALL	
ALTER TABLE .. SET TBLPROPERTIES	TABLE	ALL	
ALTER TABLE .. SET FILEFORMAT	TABLE	ALL	
ALTER TABLE .. SET LOCATION	TABLE	ALL	URI
ALTER TABLE .. ADD PARTITION	TABLE	ALL	
ALTER TABLE .. ADD PARTITION location	TABLE	ALL	URI
ALTER TABLE .. DROP PARTITION	TABLE	ALL	

Operation	Scope	Privileges Required	URI
ALTER TABLE .. PARTITION SET FILEFORMAT	TABLE	ALL	
SHOW CREATE TABLE	TABLE	SELECT/INSERT	
SHOW PARTITIONS	TABLE	SELECT/INSERT	
SHOW TABLES -Output includes all the tables for which the user has table-level privileges and all the tables for which the user has some column-level privileges.	TABLE	SELECT/INSERT	
SHOW GRANT ROLE -Output includes an additional field for any column-level privileges.	TABLE	SELECT/INSERT	
DESCRIBE TABLE -Output shows *all* columns if the user has table level-privileges or SELECT privilege on at least one table column	TABLE	SELECT/INSERT	
LOAD DATA	TABLE	INSERT	URI
SELECT -You can grant the SELECT privilege on a view to give users access to specific columns of a table they do not otherwise have access to. -See Column-level Authorization on page 381 for details on allowed column-level operations.	VIEW/TABLE; COLUMN	SELECT	
INSERT OVERWRITE TABLE	TABLE	INSERT	
CREATE TABLE .. AS SELECT -This operation is allowed if you have column-level SELECT access to the columns being used.	DATABASE; SELECT on TABLE	ALL	
USE <dbName>	Any		
CREATE FUNCTION	SERVER	ALL	
ALTER TABLE .. SET SERDEPROPERTIES	TABLE	ALL	
ALTER TABLE .. PARTITION SET SERDEPROPERTIES	TABLE	ALL	
Hive-Only Operations			
INSERT OVERWRITE DIRECTORY	TABLE	INSERT	URI

Operation	Scope	Privileges Required	URI
Analyze TABLE	TABLE	SELECT + INSERT	
IMPORT TABLE	DATABASE	ALL	URI
EXPORT TABLE	TABLE	SELECT	URI
ALTER TABLE TOUCH	TABLE	ALL	
ALTER TABLE TOUCH PARTITION	TABLE	ALL	
ALTER TABLE .. CLUSTERED BY SORTED BY	TABLE	ALL	
ALTER TABLE .. ENABLE/DISABLE	TABLE	ALL	
ALTER TABLE .. PARTITION ENABLE/DISABLE	TABLE	ALL	
ALTER TABLE .. PARTITION.. RENAME TO PARTITION	TABLE	ALL	
MSCK REPAIR TABLE	TABLE	ALL	
ALTER DATABASE	DATABASE	ALL	
DESCRIBE DATABASE	DATABASE	SELECT/INSERT	
SHOW COLUMNS -Output for this operation filters columns to which the user does not have explicit SELECT access	TABLE	SELECT/INSERT	
CREATE INDEX	TABLE	ALL	
DROP INDEX	TABLE	ALL	
SHOW INDEXES	TABLE	SELECT/INSERT	
GRANT PRIVILEGE	Allowed only for Sentry admin users		
REVOKE PRIVILEGE	Allowed only for Sentry admin users		
SHOW GRANT	Allowed only for Sentry admin users		
SHOW TBLPROPERTIES	TABLE	SELECT/INSERT	
DESCRIBE TABLE .. PARTITION	TABLE	SELECT/INSERT	
ADD JAR	Not Allowed		
ADD FILE	Not Allowed		
DFS	Not Allowed		
Impala-Only Operations			
EXPLAIN	TABLE; COLUMN	SELECT	
INVALIDATE METADATA	SERVER	ALL	
INVALIDATE METADATA <table name>	TABLE	SELECT/INSERT	

Operation	Scope	Privileges Required	URI
REFRESH <table name> or REFRESH <table name> PARTITION (<partition_spec>)	TABLE	SELECT/INSERT	
DROP FUNCTION	SERVER	ALL	
COMPUTE STATS	TABLE	ALL	

Authorization Privilege Model for Solr

The tables below refer to the request handlers defined in the generated `solrconfig.xml.secure`. If you are not using this configuration file, the below may not apply.

`admin` is a special collection in sentry used to represent administrative actions. A non-administrative request may only require privileges on the collection or config on which the request is being performed. This is called either `collection1` or `config1` in this appendix. An administrative request may require privileges on both the `admin` collection and `collection1`. This is denoted as `admin`, `collection1` in the tables below.

Note: If no privileges are granted, no access is possible. For example, accessing the Solr Admin UI requires the QUERY privilege. If no users are granted the QUERY privilege, no access to the Solr Admin UI is possible.

Table 34: Privilege table for non-administrative request handlers

Request Handler	Required Collection Privilege	Collections that Require Privilege
`select`	QUERY	`collection1`
`query`	QUERY	`collection1`
`get`	QUERY	`collection1`
`browse`	QUERY	`collection1`
`tvrh`	QUERY	`collection1`
`clustering`	QUERY	`collection1`
`terms`	QUERY	`collection1`
`elevate`	QUERY	`collection1`
`analysis/field`	QUERY	`collection1`
`analysis/document`	QUERY	`collection1`
`update`	UPDATE	`collection1`
`update/json`	UPDATE	`collection1`
`update/csv`	UPDATE	`collection1`

Table 35: Privilege table for collections admin actions

Collection Action	Required Collection Privilege	Collections that Require Privilege
`create`	UPDATE	`admin`, `collection1`
`delete`	UPDATE	`admin`, `collection1`
`reload`	UPDATE	`admin`, `collection1`

Collection Action	Required Collection Privilege	Collections that Require Privilege
`createAlias`	`UPDATE`	`admin`, `collection1` **Note:** `collection1` here refers to the name of the alias, not the underlying collection(s). For example, `http://YOUR-HOST:8983/solr/admin/collections?action=CREATEALIAS&name=collection1&collections=underlyingCollection`
`deleteAlias`	`UPDATE`	`admin`, `collection1` **Note:** `collection1` here refers to the name of the alias, not the underlying collection(s). For example, `http://YOUR-HOST:8983/solr/admin/collections?action=DELETEALIAS&name=collection1`
`syncShard`	`UPDATE`	`admin`, `collection1`
`splitShard`	`UPDATE`	`admin`, `collection1`
`deleteShard`	`UPDATE`	`admin`, `collection1`

Table 36: Privilege table for core admin actions

Collection Action	Required Collection Privilege	Collections that Require Privilege
`create`	`UPDATE`	`admin`, `collection1`
`rename`	`UPDATE`	`admin`, `collection1`
`load`	`UPDATE`	`admin`, `collection1`
`unload`	`UPDATE`	`admin`, `collection1`
`status`	`UPDATE`	`admin`, `collection1`
`persist`	`UPDATE`	`admin`
`reload`	`UPDATE`	`admin`, `collection1`
`swap`	`UPDATE`	`admin`, `collection1`
`mergeIndexes`	`UPDATE`	`admin`, `collection1`
`split`	`UPDATE`	`admin`, `collection1`
`prepRecover`	`UPDATE`	`admin`, `collection1`
`requestRecover`	`UPDATE`	`admin`, `collection1`
`requestSyncShard`	`UPDATE`	`admin`, `collection1`
`requestApplyUpdates`	`UPDATE`	`admin`, `collection1`

Table 37: Privilege table for Info and AdminHandlers

Request Handler	Required Collection Privilege	Collections that Require Privilege
LukeRequestHandler	QUERY	admin
SystemInfoHandler	QUERY	admin
SolrInfoMBeanHandler	QUERY	admin
PluginInfoHandler	QUERY	admin
ThreadDumpHandler	QUERY	admin
PropertiesRequestHandler	QUERY	admin
LogginHandler	QUERY, UPDATE (or *)	admin
ShowFileRequestHandler	QUERY	admin

Table 38: Privilege table for Config Admin actions

Config Action	Required Collection Privilege	Collections that Require Privilege	Required Config Privilege	Configs that Require Privilege
CREATE	UPDATE	admin	*	config1
DELETE	UPDATE	admin	*	config1

Before You Install Sentry

Before you install Sentry, verify the prerequisites and performance guidelines.

Prerequisites

Before you install Sentry, verify the following prerequisites:

- CDH 5.1.0 or higher managed by Cloudera Manager 5.1.0 or higher.
- If you want to configure high availability for Sentry, you must have CDH 5.13.0 or higher and Cloudera Manager 5.13.0 or higher installed.
- If you want to enable Sentry high availability, you must use a relational database, not a flat file, for the Sentry service database.
- You must have a Java version installed that has JDK-8055949 fixed.
- HiveServer2 and the Hive Metastore running with strong authentication. For HiveServer2, strong authentication is either Kerberos or LDAP. For the Hive Metastore, only Kerberos is considered strong authentication (to override, see Securing the Hive Metastore on page 375).
- If you want to use Sentry with Impala, you must have Impala 1.4.0 or higher running with strong authentication. With Impala, either Kerberos or LDAP can be configured to achieve strong authentication.
- If you want to use Sentry with Cloudera Search, the Sentry service must be configured with a database. You must have Cloudera Search for CDH 5.1.0 or higher installed. Solr supports using Sentry beginning with CDH 5.1.0. The following features were added at different releases:
  - Sentry with policy files was added in CDH 5.1.0. Note that you cannot configure Sentry high availability with policy files because high availability requires Sentry to use a relational database.
  - Sentry with config support was added in CDH 5.5.0.
  - Sentry with a relational database-backed Sentry service was added with CDH 5.8.0. If you want to use high availability for Sentry with Solr, you must use this version of Solr or higher because Sentry must be configured with a relational database.
- Implement Kerberos authentication on your cluster. For instructions, see Enabling Kerberos Authentication Using the Wizard on page 50.

Performance Guidelines

Use the following guidelines for optimal performance:

- Creating a large number of roles in Sentry can slow communication with the Hive metastore. Use 5,000 or fewer roles for best performance.
- Cloudera recommends that for each Sentry host, you have 2.5 GB memory per million objects in the Hive database.

  The amount of memory required for Sentry increases linearly as the number of objects in the Hive database increases. The graph below shows the memory required for Sentry based on the number of Hive objects.

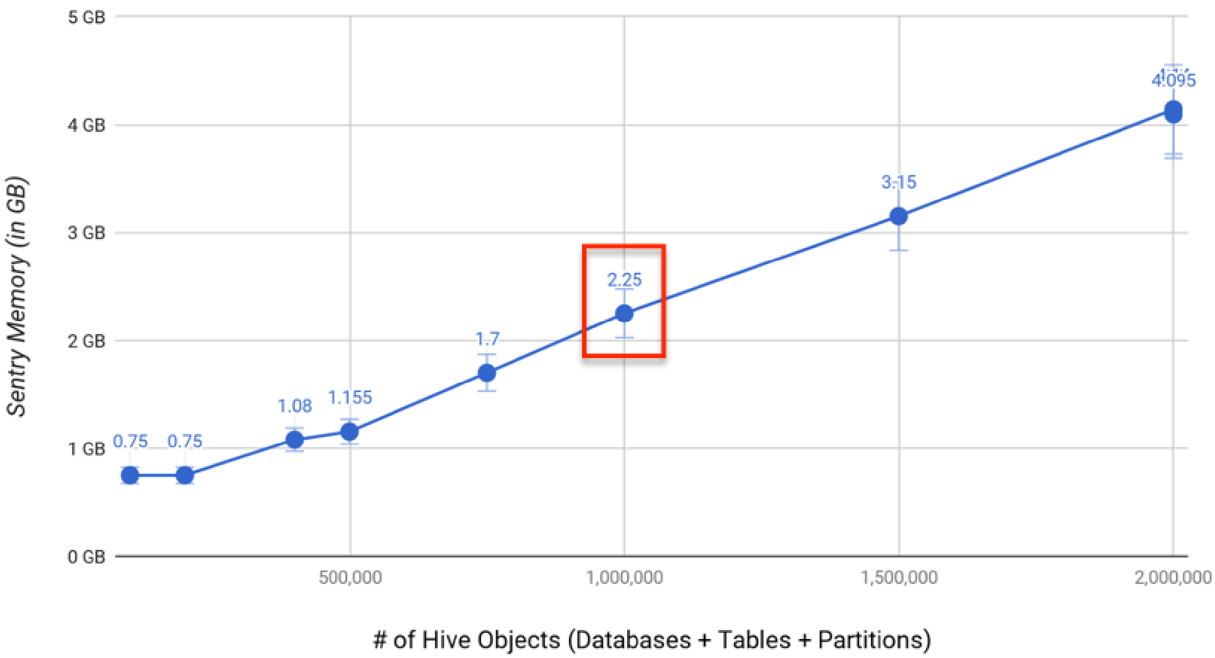


Figure 1: Sentry Memory Usage Based on Hive Objects

Installing and Upgrading the Sentry Service

This topic describes how to install and upgrade the Sentry service. If you are migrating from Sentry policy files to the database-backed Sentry service, see Migrating from Sentry Policy Files to the Sentry Service on page 363.

Before you install or upgrade the Sentry service, see the Before You Install Sentry on page 359 page to verify the prerequisites.

Adding the Sentry Service

Use one of the following sections to add/install the Sentry service:

Adding the Sentry Service Using Cloudera Manager

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

1. On the Cloudera Manager home page, click the down arrow to the right of the cluster name and select **Add a Service**. The **Add Service to Cluster** wizard opens and displays a list of service types. You can add one service at a time.
2. Select the **Sentry** service and click **Continue**.
3. The next page in the wizard allows you to customize the role assignments for Sentry. The wizard evaluates the hardware configurations of the available hosts and selects the best hosts for each role.

   If you are happy with the preselected hosts, click **Continue**. Otherwise, you can change the host by clicking the hostname.

When you click the hostname, a pop-up window appears with a list of hosts that you can choose from. You can search for a host or you can filter the list by entering a range of hosts in the **Search** field.

> **Note:** You can only select one host for the Sentry Server. After you add the Sentry service, you can add an additional host when you enable high availability. For details about how to enable high availability, see Sentry High Availability on page 375.

You can search for hosts with the following information:

- Range of hostnames. Use the following range definitions:

Range Definition	Matching Hosts
10.1.1.[1-4]	10.1.1.1, 10.1.1.2, 10.1.1.3, 10.1.1.4
host[1-3].company.com	host1.company.com, host2.company.com, host3.company.com
host[07-10].company.com	host07.company.com, host08.company.com, host09.company.com, host10.company.com

- IP addresses
- Rack name

4. After you have selected the host, click **OK** to close the window. The hostname that you selected appears under the role.
5. Click **Continue**.
6. On the **Database Setup** page, configure the database settings.

   Enter the database host, database type, database name, username, and password for the database that you created when you set up the database. See the Creating Databases documentation for Sentry Server database requirements.

7. Click **Test Connection** to confirm that Cloudera Manager can communicate with the database using the information you have supplied. If the test succeeds in all cases, click **Continue**. Otherwise check and correct the information you have provided for the database and then try the test again. (For some servers, if you are using the embedded database, you will see a message saying the database will be created at a later step in the installation process.) The Review Changes page displays.
8. Click **Continue** then click **Finish**. You are returned to the Home page.
9. Verify the new service is started properly by checking the health status for the new service. If the Health Status is **Good**, then the service started properly.
10. To use the Sentry service, begin by enabling Hive Impala, and Search for the service. See Sentry High Availability on page 375 for information about enabling high availability for Sentry.

Installing Sentry Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

Use the following the instructions, depending on your operating system, to install the latest version of Sentry.

> **Important: Configuration files**
>
> - If you install a newer version of a package that is already on the system, configuration files that you have modified will remain intact.
> - If you uninstall a package, the package manager renames any configuration files you have modified from `<file>` to `<file>.rpmsave`. If you then re-install the package (probably to install a new version) the package manager creates a new `<file>` with applicable defaults. You are responsible for applying any changes captured in the original configuration file to the new configuration file. In the case of Ubuntu and Debian upgrades, you will be prompted if you have made changes to a file for which there is a new version. For details, see [Automatic handling of configuration files by dpkg](#).

OS	Command
RHEL	`$ sudo yum install sentry`
SLES	`$ sudo zypper install sentry`
Ubuntu or Debian	`$ sudo apt-get update;` `$ sudo apt-get install sentry`

Starting the Sentry Service

Perform the following steps to start the Sentry service on your cluster.

1. Set the SENTRY_HOME and HADOOP_HOME parameters.
2. Create the Sentry database schema using the Sentry schematool. Sentry, by default, does not initialize the schema. The schematool is a built-in way for you to deploy the backend schema required by the Sentry service. For example, the following command uses the schematool to initialize the schema for a MySQL database.

```
bin/sentry --command schema-tool --conffile <sentry-site.xml> --dbType mysql --initSchema
```

   Alternatively, you can set the `sentry.verify.schema.version` configuration property to `false`. However, this is not recommended.
3. Start the Sentry service.

```
bin/sentry --command service --conffile <sentry-site.xml>
```

Upgrading the Sentry Service

Use one of the following sections to upgrade the Sentry service:

Upgrading the Sentry Service Using Cloudera Manager

If you have a cluster managed by Cloudera Manager, go to Upgrading CDH and Managed Services Using Cloudera Manager and follow the instructions depending on the version of CDH you are upgrading to. If you are upgrading from CDH 5.1, you will notice an extra step in the procedure to upgrade the Sentry database schema.

Upgrading the Sentry Service Using the Command Line

1. Stop the Sentry service by identifying the PID of the Sentry Service and use the `kill` command to end the process:

```
ps -ef | grep sentry
kill -9 <PID>
```

   Replace `<PID>` with the PID of the Sentry Service.
2. Remove the previous version of Sentry.

OS	Command
RHEL	`$ sudo yum remove sentry`
SLES	`$ sudo zypper remove sentry`
Ubuntu or Debian	`$ sudo apt-get remove sentry`

3. Install the new version of Sentry.

OS	Command
RHEL	`$ sudo yum install sentry`
SLES	`$ sudo zypper install sentry`
Ubuntu or Debian	`$ sudo apt-get update;` `$ sudo apt-get install sentry`

4. **(From CDH 5.1 to CDH 5.x) Upgrade Sentry Database Schema**

   Use the Sentry `schematool` to upgrade the database schema as follows:

```
bin/sentry --command schema-tool --conffile <sentry-site.xml> --dbType <db-type>
--upgradeSchema
```

   Where `<db-type>` should be either `mysql`, `postgres` or `oracle`.

5. **Start the Sentry Service**

   a. Set the `SENTRY_HOME` and `HADOOP_HOME` parameters.
   b. Run the following command:

```
bin/sentry --command service --conffile <sentry-site.xml>
```

6. See Sentry High Availability on page 375 for information about enabling high availability for Sentry.

Migrating from Sentry Policy Files to the Sentry Service

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

The following steps describe how you can upgrade from Sentry's policy file-based approach to the new database-backed Sentry service.

1. If you haven't already done so, upgrade your cluster to the latest version of CDH and Cloudera Manager. Refer the Cloudera Manager Administration Guide for instructions.
2. Disable the existing Sentry policy file for any Hive, Impala, or Solr services on the cluster. To do this:
   a. Go to the Hive, Impala, or Solr service.
   b. Click the **Configuration** tab.
   c. Select **Scope** > ***Service Name*** **(Service-Wide)**.
   d. Select **Category** > **Policy File Based Sentry**.
   e. Clear **Enable Sentry Authorization using Policy Files**. Cloudera Manager throws a validation error if you attempt to configure the Sentry service while this property is checked.
   f. Repeat for any remaining Hive, Impala, or Solr services.
3. Add the new Sentry service to your cluster. For instructions, see Adding the Sentry Service on page 360.
4. To begin using the Sentry service, see Enabling the Sentry Service Using Cloudera Manager on page 364 and Configuring Impala as a Client for the Sentry Service on page 371.
5. (Optional) Use command line tools to migrate existing policy file grants.

- If you want to migrate existing Sentry configurations for Solr, use the `solrctl sentry --convert-policy-file` command, described in solrctl Reference.
- For Hive and Impala, use the command-line interface Beeline to issue grants to the Sentry service to match the contents of your old policy file(s). For more details on the Sentry service and examples on using Grant/Revoke statements to match your policy file, see Hive SQL Syntax for Use with Sentry on page 381.

6. Restart the affected services as described in Restarting Services and Instances after Configuration Changes to apply the changes.

Configuring the Sentry Service

This topic describes how to enable the Sentry service for Hive and Impala, and configuring the Hive metastore to communicate with the Sentry service.

Enabling the Sentry Service Using Cloudera Manager

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

Before Enabling the Sentry Service

- Verify the prerequisites for the Sentry service: Before You Install Sentry on page 359
- Setting Hive Warehouse Directory Permissions

> **Important:** If you are going to enable HDFS/Sentry synchronization, you do not need to perform the following step to explicitly set permissions for the Hive warehouse directory. With synchronization enabled, all Hive databases and tables will automatically be owned by `hive:hive`, and Sentry permissions on tables are translated to HDFS ACLs for the underlying table files.

The Hive warehouse directory (`/user/hive/warehouse` or any path you specify as `hive.metastore.warehouse.dir` in your `hive-site.xml`) must be owned by the Hive user and group.

- **Using the default Hive warehouse directory** - Permissions on the warehouse directory must be set as follows (see following Note for caveats):
  - **771** on the directory itself (by default, `/user/hive/warehouse`)
  - **771** on all subdirectories (for example, `/user/hive/warehouse/mysubdir`)
  - All files and subdirectories should be owned by hive:hive

  For example:

```
$ sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

  If you have enabled Kerberos on your cluster, you must kinit as the `hdfs` user before you set permissions. For example:

```
sudo -u hdfs kinit -kt <hdfs.keytab> hdfs
sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

- **Using a non-default Hive warehouse:** If you would like to use a different directory as the Hive warehouse, update the `hive.metastore.warehouse.dir` property, and make sure you set the required permissions on the new directory. For example, if the new warehouse directory is `/data`, set the permissions as follows:

```
$ hdfs dfs -chown hive:hive /data
$ hdfs dfs -chmod 771 /data
```

  Note that when you update the default Hive warehouse, previously created tables will not be moved over automatically. Therefore, tables created before the update will remain at

/user/hive/warehouse/<old_table>. However, after the update, any new tables created in the default location will be found at /data/<new_table>.

For Sentry/HDFS sync to work as expected, add the new warehouse URL to the list of **Sentry Synchronization Path Prefixes**.

> **Note:**
>
> - If you set `hive.warehouse.subdir.inherit.perms` to `true` in `hive-site.xml`, the permissions on the subdirectories will be set when you set permissions on the warehouse directory.
> - If a user has access to any object in the warehouse, that user will be able to execute `use default`. This ensures that `use default` commands issued by legacy applications work when Sentry is enabled.
> - The instructions described above for modifying permissions on the Hive warehouse directory override the recommendations in the Hive section of the CDH 5 Installation Guide.

- Disable impersonation for HiveServer2 in the Cloudera Manager Admin Console. Impersonation in HiveServer2 allows users to execute queries and access HDFS files as the connected user rather than the super user who started the HiveServer2 daemon. This enforces an access control policy at the file level using HDFS file permissions or ACLs. Keeping impersonation enabled means Sentry does not have end-to-end control over the authorization process. While Sentry can enforce access control policies on tables and views in the Hive warehouse, it has no control over permissions on the underlying table files in HDFS. Hence, even if users do not have the Sentry privileges required to access a table in the warehouse, as long as they have permission to access the corresponding table file in HDFS, any jobs or queries submitted will bypass Sentry authorization checks and execute successfully. Use the following instructions to disable impersonation:

  1. Go to the Hive service.
  2. Click the **Configuration** tab.
  3. Select **Scope** > **HiveServer2**.
  4. Select **Category** > **Main**.
  5. Uncheck the **HiveServer2 Enable Impersonation** checkbox.
  6. Click **Save Changes** to commit the changes.

- If you are using MapReduce, enable the Hive user to submit MapReduce jobs.

  1. Open the Cloudera Manager Admin Console and go to the MapReduce service.
  2. Click the **Configuration** tab.
  3. Select **Scope** > **TaskTracker**.
  4. Select **Category** > **Security**.
  5. Set the **Minimum User ID for Job Submission** property to zero (the default is 1000).
  6. Click **Save Changes** to commit the changes.
  7. Repeat steps 1-6 for *every* TaskTracker role group for the MapReduce service that is associated with Hive.
  8. Restart the MapReduce service.

- If you are using YARN, enable the Hive user to submit YARN jobs.

  1. Open the Cloudera Manager Admin Console and go to the YARN service.
  2. Click the **Configuration** tab.
  3. Select **Scope** > **NodeManager**.
  4. Select **Category** > **Security**.
  5. Ensure the **Allowed System Users** property includes the `hive` user. If not, add `hive`.
  6. Click **Save Changes** to commit the changes.
  7. Repeat steps 1-6 for *every* NodeManager role group for the YARN service that is associated with Hive.
  8. Restart the YARN service.

- Block the Hive CLI user from accessing the Hive metastore:
  1. In the Cloudera Manager Admin Console, select the **Hive** service.
  2. On the Hive service page, click the **Configuration** tab.
  3. In the search well on the right half of the Configuration page, search for `Hive Metastore Access Control and Proxy User Groups Override` to locate the `hadoop.proxyuser.hive.groups` parameter and click the plus sign.
  4. Enter `hive` into the text box and click the plus sign again.
  5. Enter hue into the text box.
  6. Click **Save Changes**.

  Setting this parameter blocks access to the Hive metastore for the user running the Hive CLI if they are not part of the `hive` or the hue groups. The Hive CLI can still run, but after setting this parameter as described here, the `hive` user can impersonate only members of the `hive` or the hue groups. If you are using Sqoop, the Sqoop user must also have access to the Hive metastore.

> **Important:** Ensure you have unchecked the **Enable Sentry Authorization using Policy Files** configuration property for *both* Hive and Impala under the **Policy File Based Sentry** category before you proceed.

Enabling the Sentry Service for Hive

1. Go to the Hive service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Hive (Service-Wide)**.
4. Select **Category** > **Main**.
5. Locate the **Sentry Service** property and select `Sentry`.
6. Locate the **Enable Stored Notifications in Database** property and select it.
7. Click **Save Changes** to commit the changes.
8. Restart the Hive service.

Enabling Sentry on Hive service places several HiveServer2 properties on a restricted list properties that cannot be modified at runtime by clients. See HiveServer2 Restricted Properties on page 374.

Enabling the Sentry Service for Impala

1. Enable the Sentry service for Hive (as instructed above).
2. Go to the Impala service.
3. Click the **Configuration** tab.
4. Select **Scope** > **Impala (Service-Wide)**.
5. Select **Category** > **Main**.
6. Locate the **Sentry Service** property and select `Sentry`.
7. Click **Save Changes** to commit the changes.
8. Restart Impala.

Enabling the Sentry Service for Solr

Enable the Sentry service as follows:

1. Go to the Solr service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Solr (Service-Wide)**.
4. Select **Category** > **Main**.
5. Locate the **Sentry Service** property and select `Sentry`.
6. Click **Save Changes** to commit the changes.

7. Restart Solr.

After enabling Sentry for Solr, you may want to configure authorization as described in Configuring Sentry Authorization for Cloudera Search on page 425.

Enabling the Sentry Service for Hue

Hue uses a Security app to make it easier to interact with Sentry. When you set up Hue to manage Sentry permissions, make sure that users and groups are set up correctly. Every Hue user connecting to Sentry must have an equivalent OS-level user account on all hosts so that Sentry can authenticate Hue users. Each OS-level user should also be part of an OS-level group with the same name as the corresponding user's group in Hue.

For more information on using the Security app, see the related blog post.

Enable the Sentry service as follows:

1. Enable the Sentry service for Hive and Impala (as instructed above).
2. Go to the Hue service.
3. Click the **Configuration** tab.
4. Select **Scope** > **Hue (Service-Wide)**.
5. Select **Category** > **Main**.
6. Locate the **Sentry Service** property and select `Sentry`.
7. Click **Save Changes** to commit the changes.
8. Restart Hue.

Add the Hive, Impala and Hue Groups to Sentry's Admin Groups

1. Go to the Sentry service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Sentry (Service-Wide)**.
4. Select **Category** > **Main**.
5. Locate the **Admin Groups** property and add the `hive`, `impala` and hue groups to the list. If an end user is in one of these admin groups, that user has administrative privileges on the Sentry Server.
6. Click **Save Changes** to commit the changes.

Enabling the Sentry Service Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

Before Enabling the Sentry Service

- Setting Hive Warehouse Directory Permissions

  > **Important:** If you are going to enable HDFS/Sentry synchronization, you do not need to perform the following step to explicitly set permissions for the Hive warehouse directory. With synchronization enabled, all Hive databases and tables will automatically be owned by `hive:hive`, and Sentry permissions on tables are translated to HDFS ACLs for the underlying table files.

  The Hive warehouse directory (`/user/hive/warehouse` or any path you specify as `hive.metastore.warehouse.dir` in your `hive-site.xml`) must be owned by the Hive user and group.

  – **Using the default Hive warehouse directory** - Permissions on the warehouse directory must be set as follows (see following Note for caveats):

- **771** on the directory itself (by default, `/user/hive/warehouse`)
- **771** on all subdirectories (for example, `/user/hive/warehouse/mysubdir`)
- All files and subdirectories should be owned by hive:hive

For example:

```
$ sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

If you have enabled Kerberos on your cluster, you must kinit as the `hdfs` user before you set permissions. For example:

```
sudo -u hdfs kinit -kt <hdfs.keytab> hdfs
sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

- **Using a non-default Hive warehouse:** If you would like to use a different directory as the Hive warehouse, update the `hive.metastore.warehouse.dir` property, and make sure you set the required permissions on the new directory. For example, if the new warehouse directory is `/data`, set the permissions as follows:

```
$ hdfs dfs -chown hive:hive /data
$ hdfs dfs -chmod 771 /data
```

Note that when you update the default Hive warehouse, previously created tables will not be moved over automatically. Therefore, tables created before the update will remain at `/user/hive/warehouse/<old_table>`. However, after the update, any new tables created in the default location will be found at `/data/<new_table>`.

For Sentry/HDFS sync to work as expected, add the new warehouse URL to the list of **Sentry Synchronization Path Prefixes**.

Note:

- If you set `hive.warehouse.subdir.inherit.perms` to `true` in `hive-site.xml`, the permissions on the subdirectories will be set when you set permissions on the warehouse directory.
- If a user has access to any object in the warehouse, that user will be able to execute `use default`. This ensures that `use default` commands issued by legacy applications work when Sentry is enabled.
- The instructions described above for modifying permissions on the Hive warehouse directory override the recommendations in the Hive section of the CDH 5 Installation Guide.

- HiveServer2 impersonation must be turned off. Impersonation in HiveServer2 allows users to execute queries and access HDFS files as the connected user rather than the super user who started the HiveServer2 daemon. This enforces an access control policy at the file level using HDFS file permissions or ACLs. Keeping impersonation enabled means Sentry does not have end-to-end control over the authorization process. While Sentry can enforce access control policies on tables and views in the Hive warehouse, it has no control over permissions on the underlying table files in HDFS. Hence, even if users do not have the Sentry privileges required to access a table in the warehouse, as long as they have permission to access the corresponding table file in HDFS, any jobs or queries submitted will bypass Sentry authorization checks and execute successfully.
- If you are using MapReduce, you must enable the Hive user to submit MapReduce jobs. You can ensure that this is true by setting the minimum user ID for job submission to 0. Edit the `taskcontroller.cfg` file and set `min.user.id=0`.

If you are using YARN, you must enable the Hive user to submit YARN jobs, add the user `hive` to the `allowed.system.users` configuration property. Edit the `container-executor.cfg` file and add `hive` to the `allowed.system.users` property. For example,

```
allowed.system.users = nobody,impala,hive,hbase
```

> **Important:** You must restart the cluster and HiveServer2 after changing these values.

- Block the Hive CLI user from accessing the Hive metastore by setting the following property in the cluster's `core-site.xml` file:

```
<property>
     <name>hadoop.proxyuser.hive.groups</name>
     <value>hive,hue</value>
     <description>Sets groups from which the hive user can impersonate other
users.</description>
</property>
```

Setting this parameter blocks access to the Hive metastore for the user running the Hive CLI if they are not part of the `hive` or the hue groups. The Hive CLI can still run, but after setting this parameter as described here, the `hive` user can impersonate only members of the `hive` or the hue groups. If you are using Sqoop, the Sqoop user must also have access to the Hive metastore.

- Add the `hive`, `impala` and hue groups to Sentry's `sentry.service.admin.group` in the `sentry-site.xml` file. If an end user is in one of these admin groups, that user has administrative privileges on the Sentry Server.

```
<property>
  <name>sentry.service.admin.group</name>
  <value>hive,impala,hue</value>
 </property>
```

Configuring the Sentry Server

Configure the following properties in `sentry-site.xml` on the Sentry Server host.

```
<property>
    <name>sentry.service.server.rpc-address</name>
    <value>nightly54-1.gce.cloudera.com</value>
</property>
  <property>
    <name>sentry.service.server.rpc-port</name>
    <value>8038</value>
</property>
<property>
    <name>sentry.service.admin.group</name>
    <value>hive,impala,hue</value>
</property>
<property>
    <name>sentry.service.allow.connect</name>
    <value>hive,impala,hue,hdfs</value>
</property>
<property>
    <name>sentry.store.group.mapping</name>
    <value>org.apache.sentry.provider.common.HadoopGroupMappingService</value>
</property>
<property>
    <name>sentry.service.server.principal</name>
    <value>sentry/_HOST@GCE.CLOUDERA.COM</value>
</property>
<property>
    <name>sentry.service.security.mode</name>
    <value>kerberos</value>
```

```
</property>
<property>
    <name>sentry.service.server.keytab</name>
    <value>sentry.keytab</value>
</property>
<property>
    <name>sentry.store.jdbc.url</name>
    <value>jdbc:<JDBC connection URL for backend database></value>
</property>
<property>
    <name>sentry.store.jdbc.driver</name>
    <value><JDBC Driver class for backend database></value>
</property>
<property>
    <name>sentry.store.jdbc.user</name>
    <value><User ID for backend database user></value>
</property>
<property>
    <name>sentry.store.jdbc.password</name>
    <value><Password for backend database user></value>
</property>
<property>
    <name>sentry.service.processor.factories</name>
   <value>org.apache.sentry.provider.db.service.thrift.SentryPolicyStoreProcessorFactory,

             org.apache.sentry.hdfs.SentryHDFSServiceProcessorFactory</value>
</property>
<property>
    <name>sentry.policy.store.plugins</name>
    <value>org.apache.sentry.hdfs.SentryPlugin</value>
</property>
<property>
    <name>sentry.hdfs.integration.path.prefixes</name>
    <value>/user/hive/warehouse</value>
</property>
```

Configuring HiveServer2 for the Sentry Service

Configure the following properties in `sentry-site.xml` on the HiveServer2 host.

```
<property>
    <name>hive.sentry.server</name>
    <value>server1</value>
</property>
<property>
    <name>sentry.service.server.principal</name>
    <value>sentry/_HOST@EXAMPLE.COM</value>
</property>
<property>
    <name>sentry.service.security.mode</name>
    <value>kerberos</value>
</property>
<property>
    <name>sentry.hive.provider.backend</name>
    <value>org.apache.sentry.provider.db.SimpleDBProviderBackend</value>
</property>
<property>
    <name>sentry.service.client.server.rpc-address</name>
    <value>example.cloudera.com</value>
</property>
<property>
    <name>sentry.service.client.server.rpc-port</name>
    <value>8038</value>
</property>
<property>
    <name>hive.sentry.provider</name>

<value>org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider</value>
</property>
<property>
    <name>hive.sentry.failure.hooks</name>
```

```
    <value>com.cloudera.navigator.audit.hive.HiveSentryOnFailureHook</value>
</property>
```

Add the following properties to `hive-site.xml` to allow the Hive service to communicate with the Sentry service.

```
<property>
   <name>hive.security.authorization.task.factory</name>
   <value>org.apache.sentry.binding.hive.SentryHiveAuthorizationTaskFactoryImpl</value>
</property>
<property>
   <name>hive.server2.session.hook</name>
   <value>org.apache.sentry.binding.hive.HiveAuthzBindingSessionHook</value>
</property>
<property>
   <name>hive.sentry.conf.url</name>
   <value>file:///{{PATH/TO/DIR}}/sentry-site.xml</value>
</property>
<property>
   <name>hive.security.authorization.task.factory</name>
   <value>org.apache.sentry.binding.hive.SentryHiveAuthorizationTaskFactoryImpl</value>
</property>
```

Enabling Sentry on Hive service places several HiveServer2 properties on a restricted list properties that cannot be modified at runtime by clients. See HiveServer2 Restricted Properties on page 374.

Configuring the Hive Metastore for the Sentry Service

Add the following properties to `hive-site.xml` to allow the Hive metastore to communicate with the Sentry service.

```
<property>
<name>hive.metastore.filter.hook</name>
<value>org.apache.sentry.binding.metastore.SentryMetaStoreFilterHook</value>
</property>

<property>
    <name>hive.metastore.pre.event.listeners</name>
    <value>org.apache.sentry.binding.metastore.MetastoreAuthzBinding</value>
    <description>list of comma separated listeners for metastore events.</description>
</property>

<property>
    <name>hive.metastore.event.listeners</name>
    <value>org.apache.sentry.binding.metastore.SentryMetastorePostEventListener</value>

    <description>list of comma separated listeners for metastore, post
events.</description>
</property>
```

Configuring Impala as a Client for the Sentry Service

Set the following configuration properties in `../impala-conf/sentry-site.xml` on the Catalog Server.

```
<property>
   <name>sentry.service.client.server.rpc-port</name>
   <value>8038</value>
</property>
<property>
   <name>sentry.service.client.server.rpc-address</name>
   <value>hostname</value>
</property>
<property>
   <name>sentry.service.client.server.rpc-connection-timeout</name>
   <value>200000</value>
</property>
<property>
   <name>sentry.service.security.mode</name>
   <value>kerberos</value>
</property>
```

You must also add the following configuration properties to Impala's /etc/default/impala file. For more information , see Configuring Impala Startup Options through the Command Line.

- On the catalogd and the impalad.

```
--sentry_config=<absolute path to sentry service configuration file>
```

- On the impalad.

```
--server_name=<server name>
```

If the --authorization_policy_file flag is set, Impala will use the policy file-based approach. Otherwise, the database-backed approach will be used to implement authorization.

Enabling Solr as a Client for the Sentry Service Using the Command Line

You can enable Sentry using Cloudera Manager or by manually modifying files. For more information on enabling Sentry using Cloudera Manager, see Configuring Sentry Policy File Authorization Using Cloudera Manager on page 406 and Enabling Sentry Policy File Authorization for Solr on page 410.

Sentry is enabled with addition of two properties to /etc/default/solr or /opt/cloudera/parcels/CDH-*/etc/default/solr.

- In a Cloudera Manager deployment, required properties are added automatically when you click **Enable Sentry Authorization** in the Solr configuration page in Cloudera Manager.
- If you are using configs, you must configure the proper config=myConfig permissions as described in Using Roles and Privileges with Sentry on page 426.
- In a deployment not managed by Cloudera Manager, you must make these changes yourself. The variable SOLR_AUTHORIZATION_SENTRY_SITE specifies the path to sentry-site.xml. The variable SOLR_AUTHORIZATION_SUPERUSER specifies the first part of SOLR_KERBEROS_PRINCIPAL. This is solr for the majority of users, as solr is the default. Settings are of the form:

```
SOLR_AUTHORIZATION_SENTRY_SITE=/location/to/sentry-site.xml
SOLR_AUTHORIZATION_SUPERUSER=solr
```

To enable Sentry collection-level authorization checking on a new collection, the instancedir for the collection must use a modified version of solrconfig.xml with Sentry integration. Each collection has a separate solrconfig.xml file, meaning you can define different behavior for each collection. The command solrctl instancedir --generate generates two versions of solrconfig.xml: the standard solrconfig.xml without sentry integration and the sentry-integrated version called solrconfig.xml.secure. To use the sentry-integrated version, replace solrconfig.xml with solrconfig.xml.secure before creating the instancedir.

You can enable Sentry on an existing collection. The process varies depending on whether you are using a config or instancedir.

Enabling Sentry on Collections using configs

If you have a collection that is using a non-secured config, you can enable Sentry security on that collection by modifying the collection to use a secure config. The config in use must not be immutable, otherwise it cannot be changed. To update an existing non-immutable config:

1. Delete the existing config using the solrctl config --delete command. For example:

```
solrctl config --delete myManaged
```

2. Create a new non-immutable config using the solrctl config --create command. Use a sentry-enabled template such as managedTemplateSecure. The new config must have the same name as the config being replaced. For example:

```
solrctl config --create myManaged managedTemplateSecure -p immutable=false
```

3. Reload the collection using to `solrctl collection --reload` command.

```
solrctl collection --reload myCollection
```

For a list of all available config templates, see Config Templates.

Enabling Sentry on Collections using instancedirs

If you have a collection that is using a non-secured instancedir configuration, you can enable Sentry security on that collection by modifying the settings that are stored in `instancedir`. For example, you might have an existing collection named `foo` and a standard `solrconfig.xml`. By default, collections are stored in instancedirs that use the collection's name, which is `foo` in this case.

If your collection uses an unmodified `solrconfig.xml` file, you can enable Sentry by replacing the existing `solrconfig.xml` file. If your collection uses a `solrconfig.xml` that contains modifications you want to preserve, you can attempt to use a difftool to find an integrate changes in to the secure template.

To enable Sentry on an existing collection without preserving customizations

Warning: Running the following commands replaces your existing `solrconfig.xml` file. Any customizations to this file will be lost.

```
# generate a fresh instancedir
solrctl instancedir --generate foosecure
# download the existing instancedir from ZK into subdirectory foo
solrctl instancedir --get foo foo
# replace the existing solrconfig.xml with the sentry-enabled one
cp foosecure/conf/solrconfig.xml.secure foo/conf/solrconfig.xml
# update the instancedir in ZK
solrctl instancedir --update foo foo
# reload the collection
solrctl collection --reload foo
```

To enable Sentry on an existing collection and preserve customizations

Generate a new instancedir, compare the differences between the default `solrconfig.xml` and `solrconfig.xml.secure` files, and then add the elements that are unique to `solrconfig.xml.secure` to the file that your environment is using.

1. Generate a fresh instancedir:

```
solrctl instancedir --generate foo
```

2. Compare the `solrconfig.xml` and `solrconfig.xml.secure`:

```
diff foo/conf/solrconfig.xml foo/conf/solrconfig.xml.secure
```

3. Add the elements that are unique to `solrconfig.xml.secure` to your existing `solrconfig.xml` file. You might complete this process by manually editing your existing `solrconfig.xml` file or by using a merge tool.

> **Note:** If you have modified or specified additional request handlers, consider that Sentry:
>
> - Supports protecting additional query request handlers by adding a search component, which should be shown in the diff.
> - Supports protecting additional update request handlers with Sentry by adding an updateRequestProcessorChain, which should be shown in the diff.
> - Does not support protecting modified or specified additional "special" request handlers like analysis handlers or admin handlers.

4. Reload the collection:

```
solrctl collection --reload foo
```

After enabling Sentry for Solr, you may want to configure authorization as described in Configuring Sentry Authorization for Cloudera Search on page 425.

HiveServer2 Restricted Properties

Enabling Sentry on Hive service places several HiveServer2 properties on a restricted list properties that cannot be modified at runtime by clients. This list is denoted by the `hive.conf.restricted.list` property and these properties are only configurable on the server side. The list includes:

```
hive.enable.spark.execution.engine
hive.semantic.analyzer.hook
hive.exec.pre.hooks
hive.exec.scratchdir
hive.exec.local.scratchdir
hive.metastore.uris,
javax.jdo.option.ConnectionURL
hadoop.bin.path
hive.session.id
hive.aux.jars.path
hive.stats.dbconnectionstring
hive.scratch.dir.permission
hive.security.command.whitelist
hive.security.authorization.task.factory
hive.entity.capture.transform
hive.access.conf.url
hive.sentry.conf.url
hive.access.subject.name
hive.sentry.subject.name
hive.sentry.active.role.set
```

Configuring Pig and HCatalog for the Sentry Service

Once you have the Sentry service up and running, and Hive has been configured to use the Sentry service, there are some configuration changes you must make to your cluster to allow Pig, MapReduce (using HCatLoader, HCatStorer) and WebHCat queries to access Sentry-secured data stored in Hive.

Since the Hive warehouse directory is owned by `hive:hive`, with its permissions set to **`771`**, with these settings, other user requests such as commands coming through Pig jobs, WebHCat queries, and MapReduce jobs, may fail. To give these users access, perform the following configuration changes:

- Use HDFS ACLs to define permissions on a specific directory or file of HDFS. This directory/file is generally mapped to a database, table, partition, or a data file.
- Users running these jobs should have the required permissions in Sentry to add new metadata or read metadata from the Hive Metastore Server. For instructions on how to set up the required permissions, see Hive SQL Syntax for Use with Sentry on page 381. You can use HiveServer2's command line interface, Beeline to update the Sentry database with the user privileges.

Examples:

- A user who is using Pig HCatLoader will require read permissions on a specific table or partition. In such a case, you can GRANT read access to the user in Sentry and set the ACL to read and run, on the file being accessed.
- A user who is using Pig HCatStorer will require ALL permissions on a specific table. In this case, you GRANT ALL access to the user in Sentry and set the ACL to write and run on the table being used.

Securing the Hive Metastore

It's important that the Hive metastore be secured. If you want to override the Kerberos prerequisite for the Hive metastore, set the `sentry.hive.testing.mode` property to `true` to allow Sentry to work with weaker authentication mechanisms. Add the following property to the HiveServer2 and Hive metastore's `sentry-site.xml`:

```
<property>
  <name>sentry.hive.testing.mode</name>
  <value>true</value>
</property>
```

Impala does not require this flag to be set.

> **Warning:** Cloudera strongly recommends against enabling this property in production. Use Sentry's testing mode only in test environments.

You can turn on Hive metastore security using the instructions in Cloudera Security. To secure the Hive metastore; see Hive Metastore Server Security Configuration on page 132.

Using User-Defined Functions with HiveServer2

The `ADD JAR` command does *not* work with HiveServer2 and the Beeline client when Beeline runs on a different host. As an alternative to `ADD JAR`, Hive's *auxiliary paths* functionality should be used. There are some differences in the procedures for creating permanent functions and temporary functions when Sentry is enabled. For detailed instructions, see:

- Using Cloudera Manager to Create User-Defined Functions (UDFs) with HiveServer2

  OR

- User-Defined Functions (UDFs) with HiveServer2 Using the Command Line

.

Sentry High Availability

High availability for Sentry provides automatic failover in the event that your primary Sentry host goes down or is unavailable. In CDH 5.13, you can select one additional host for high availability, for a total of two Sentry hosts.

Architecture

With Sentry high availability enabled, there is a leader Sentry Server that gets the current Hive metadata from the Hive metastore and permissions from HiveServer2, and writes that information to the Sentry database. The other Sentry Server then asks the Sentry database for the updated information; the Sentry Servers do not communicate with each other directly. The Sentry Server leader is determined randomly by the ZooKeeper service and is not configurable by the administrator. If the Sentry Server leader goes down, the other Sentry Server becomes the leader. Load balancing between the Sentry services is achieved by each client randomly selecting a Sentry service to connect to.

The image below illustrates how the Sentry Servers interact with the clients and the Sentry database when both Sentry Servers are available. The clients can connect to both Sentry Servers. The leader Sentry Server reads from and writes to the Sentry database, and the other Sentry Server reads from the Sentry database.

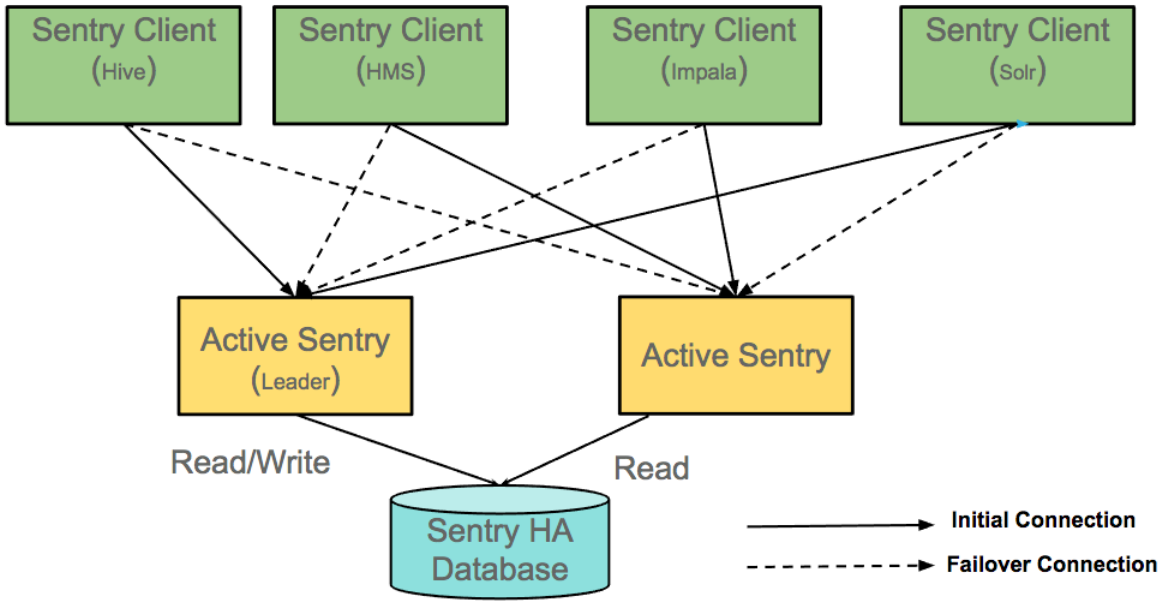


Figure 2: Sentry High Availability Architecture

When one of the Sentry Servers is unavailable, as in the following image, the clients connect to the Sentry Server that is available. As the image illustrates, if the leader Sentry Server is unavailable, the other Sentry Server can write to the Sentry database.

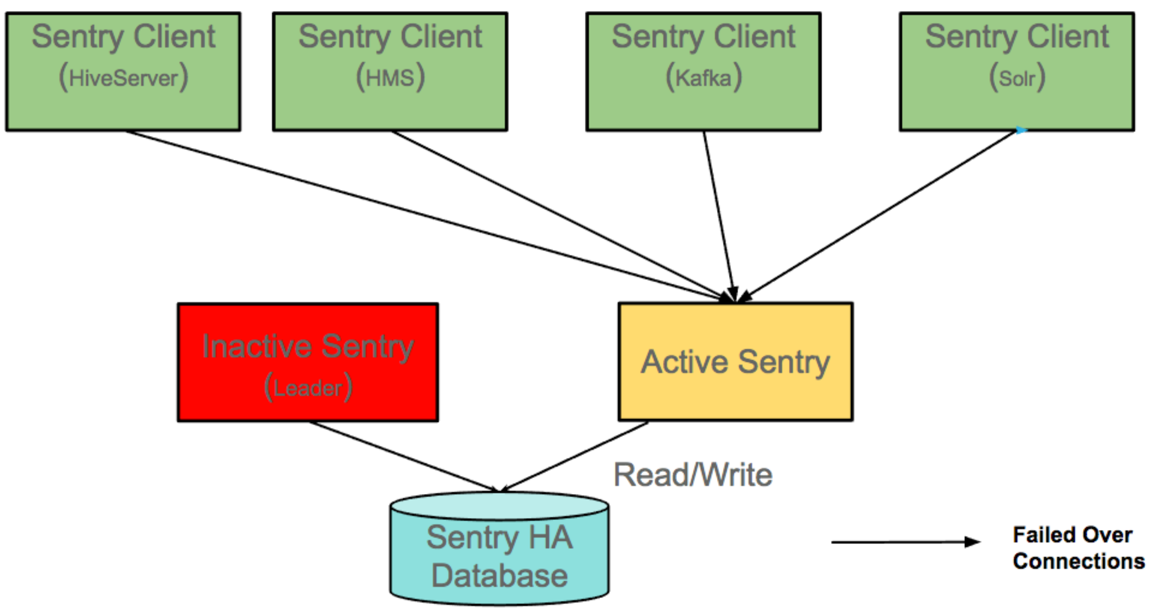


Figure 3: Sentry High Availability Architecture with Failover

Sentry and Hive Metastore High Availability

You can also use Sentry high availability with Hive metastore high availability. When all the Sentry Servers and Hive metastores are available, the Sentry leader pulls Hive metadata from the Hive metastore master. If one or more is unavailable, the available Sentry Server that is available pulls metadata from the Hive metastore that is available.

For more information about Hive metastore high availability, see [Configuring Apache Hive Metastore High Availability in CDH](#).

In the image below, one of the Sentry Servers in unavailable. The Sentry Server that is available pulls metadata from the Hive metastore leader.

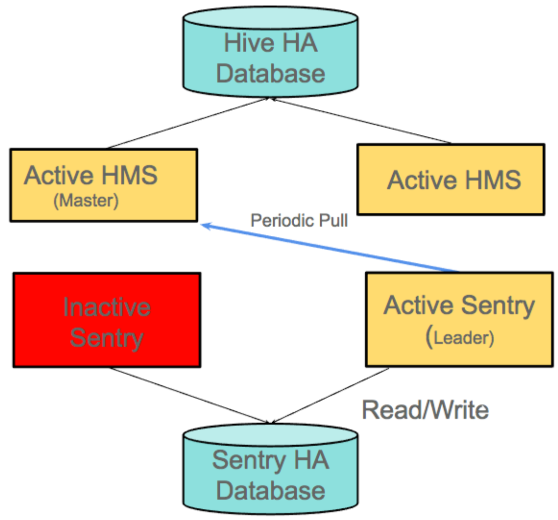


Figure 4: Sentry and HMS High Availability Architecture with Sentry Failover

When one of the Hive metastores is unavailable, the Sentry leader pulls metadata from the metastore that is available, as the following figure illustrates:

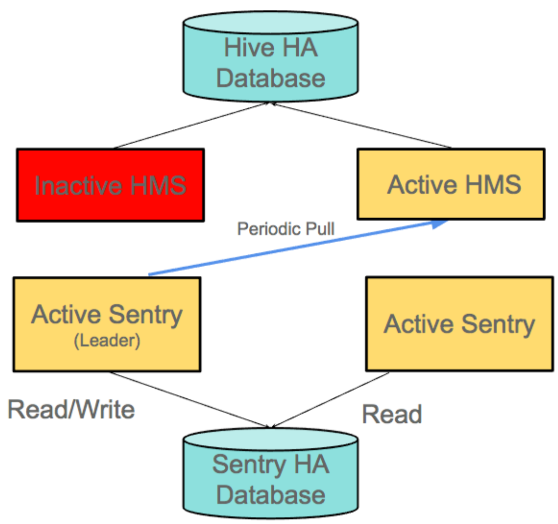


Figure 5: Sentry and HMS High Availability Architecture with HMS Failover

Prerequisites

Verify the following prerequisites before you enable high availability for Sentry:

- You must have CDH version 5.13.0 or higher installed.
- You must have Cloudera Manager version 5.13.0 or higher installed.
- The Sentry service must be installed and running. See Before You Install Sentry on page 359 and Installing and Upgrading the Sentry Service on page 360 for information about how to install the Sentry service.
- The ZooKeeper service must be enabled. To view the ZooKeeper service that the Sentry service depends on, open the Sentry service in Cloudera Manager, click the **Configuration** tab and enter **ZooKeeper** in the **Search** box. The **ZooKeeper Service** property is displayed.

- The Sentry service must be backed by a relational database, not a flat file.
- If you want to use a rolling restart to restart the cluster, you must have HDFS high availability enabled. For information about configuring HDFS high availability, see HDFS High Availability.

Enable Sentry High Availability

You can enable high availability for Sentry using a rolling restart or with a manual cluster restart. A rolling restart allows you to restart the cluster without completely bringing down the cluster, but you must have HDFS high availability enabled.

The following sections describe the steps to enable high availability with and without a rolling restart. You can also view the how-to article, How to Enable Sentry High Availability for steps to enable high availability with images to guide you through the process.

Enabling with a Rolling Restart

You can perform a rolling restart through Cloudera Manager if you want to enable Sentry high availability without bringing down the cluster. You must have HDFS high availability enabled and you must perform a cluster-level rolling restart. Note that while you can use a rolling restart to enable Sentry high availability without stopping the cluster, the restart will stop the Sentry service.

For information about configuring HDFS high availability, see HDFS High Availability.

For more information about using Cloudera Manager to perform a rolling restart, see Rolling Restart.

Complete the following steps to enable Sentry high availability with a rolling restart:

1. In Cloudera Manager, open the Sentry service.
2. Open the **Instances** tab and click **Add Role Instances**.
3. In the **Add Role Instances** wizard, click the **Select a host** button for the Sentry Server. You do not have to edit the Gateway host.
4. A pop-up window opens that allows you to pick an alternate host for the Sentry service. Click the hostname of the host that you want to use for high availability. The host that the Sentry service is currently running on is already selected and grayed out. You can select one alternate host for the Sentry service.

   You can search for a host or you can filter the list by entering a range of hosts in the **Search** field. You can search for hosts with the following information:

   - Range of hostnames. Use the following range definitions:

Range Definition	Matching Hosts
10.1.1.[1-4]	10.1.1.1, 10.1.1.2, 10.1.1.3, 10.1.1.4
host[1-3].company.com	host1.company.com, host2.company.com, host3.company.com
host[07-10].company.com	host07.company.com, host08.company.com, host09.company.com, host10.company.com

   - IP addresses
   - Rack name

5. After you have selected the alternate host, click **OK** to close the window. The hostname appears below the Sentry Server.
6. Click **Continue**. The wizard closes and the Sentry hosts are listed.
7. Go to the Cloudera Manager home page and click the **All Recent Commands** tab. Verify that the **Generate Missing Credentials** command has successfully completed before you proceed with the rolling restart.
8. Click the **Status** tab to return to the Cloudera Manager home page.
9. Click the down arrow next to the cluster name to open the menu and click **Deploy Client Configuration**.
10. A window opens asking if you are sure you want to run the Deploy Client Configuration command. Click **Deploy Client Configuration**.
11. When the steps are successfully completed, click **Close** to close the window.

12. Click the down arrow next to the cluster name again to open the menu and click **Rolling Restart**. For detailed information about the rolling restart, see Rolling Restart.
13. In the **Rolling Restart** window that opens, select the services that have stale configurations, including the Sentry service. Every service that is dependent on Sentry will have a stale configuration. In the **Roles to Include** parameter, select **All Roles**.

    Note that the services that do not support rolling restart will simply be restarted and will be unavailable during their restart, including the Sentry service.

14. Click **Rolling Restart** to begin the restart.
15. A pop-up window opens that lists the rolling restart steps and the status of each step. When the rolling restart is complete, click **Close** to close the pop-up.
16. Verify that none of the services have stale configurations. If one or more services has a stale configuration after the rolling restart, Sentry high availability might not be configured properly.
17. To verify the statuses of the Sentry hosts, open the Sentry service and click the **Instances** tab.

Enabling with a Cluster Restart

Complete the following steps to enable high availability for Sentry in Cloudera Manager with a cluster restart:

1. In Cloudera Manager, open the Sentry service.
2. In the **Actions** drop-down menu, click **Enable High Availability**.

    The wizard walks you through the steps to enable high availability.

3. Click **Select a Host** in the **Sentry Server Hosts** field. You do not have to edit the Gateway host.
4. A pop-up window opens that allows you to pick an alternate host for the Sentry service. Click the hostname of the host that you want to use for high availability. The host that the Sentry service is currently running on is already selected and grayed out. You can select one alternate host for the Sentry service.

    You can search for a host or you can filter the list by entering a range of hosts in the **Search** field. You can search for hosts with the following information:

    - Range of hostnames. Use the following range definitions:

Range Definition	Matching Hosts
10.1.1.[1-4]	10.1.1.1, 10.1.1.2, 10.1.1.3, 10.1.1.4
host[1-3].company.com	host1.company.com, host2.company.com, host3.company.com
host[07-10].company.com	host07.company.com, host08.company.com, host09.company.com, host10.company.com

    - IP addresses
    - Rack name

    Click the hostname to select the host and click **OK** to close the window.
5. The two Sentry server hosts are listed in the **Sentry Server Hosts** field. Click **Continue**.
6. You can see the status of the of the new Sentry Server and the cluster restart. When the steps have successfully completed, click **Finish** to close the wizard.
7. From the Sentry service page, you can view the status of your hosts by clicking the **Instances** tab.

Disable Sentry High Availability

You can also disable high availability for Sentry with a rolling restart or with a manual cluster restart. As with enabling high availability, the rolling restart allows you to restart the cluster with minimal downtime, but you must have HDFS high availability enabled.

Complete the steps below to disable high availability for Sentry:

Disabling with a Rolling Restart

You can use a rolling restart to disable Sentry high availability without bringing down the cluster. Although you do not have to stop the cluster, the rolling restart does stop the Sentry service.

Complete the following steps to disable high availability for Sentry with a rolling restart:

1. In Cloudera Manager, open the Sentry service.
2. Click the **Instances** tab.
3. Select the instance that you no longer want to be a host for the Sentry service.
4. Click **Actions for Selected** > **Stop**.
5. In the pop-up window that appears, click **Stop**.
6. The pop-up window shows the status of the command. When it is complete, click **Close** to close the pop-up.
7. With the host still selected, click **Actions for Selected** > **Delete**.
8. In the pop-up window that appears, click **Delete**.
9. The pop-up closes and the host is removed from the list of Sentry hosts.
10. Navigate to the Cloudera Manager home page.
11. Click the down arrow next to the cluster name to open the menu and click **Rolling Restart**. For detailed information about the rolling restart, see [Rolling Restart](#).
12. Select the services that you want to restart, including the Sentry service, and click **Rolling Restart**. Note that the services that do not support rolling restart will simply be restarted and will be unavailable during their restart, including the Sentry service.
13. A pop-up window opens that lists the rolling restart steps and the status of each step. When the rolling restart is complete, click **Close** to close the pop-up.
14. To verify the status of the Sentry host, open the Sentry service and click the **Instances** tab.

Disabling with a Cluster Restart

Complete the following steps to disable high availability for Sentry in Cloudera Manager with a cluster restart:

1. In Cloudera Manager, open the Sentry service.
2. In the **Actions** drop-down menu, click **Disable High Availability**.

   The wizard walks you through the steps to disable high availability.

3. Select the host that you want the Sentry service to run on after you disable high availability. Click the hostname and click **Continue**.
4. You can see the status of the Sentry server and cluster restart. When the steps have successfully completed, click **Finish** to close the wizard.
5. When the wizard closes, Cloudera Manager takes you back to the Sentry service page. You can view the status of your host by clicking the **Instances** tab.

Sentry Debugging and Failure Scenarios

This topic explains how Sentry deals with policy conflicts and how different CDH components respond when the Sentry service fails. It also shows you how to debug Sentry authorization request failures. Always first review the CDH Release Notes and the list of [Known Issues for Sentry](#) when you have issues.

Resolving Policy Conflicts

Sentry treats all policies independently. That means that for any operation, if Sentry finds an policy that allows it, that operation will be allowed. Consider an example with a table, `test_db.test_tbl`, whose HDFS directory is located at `hdfs://user/hive/warehouse/test_db.db/test_tbl`, and grant the following conflicting privileges to a user with the role, `test_role`. That is, you are granting ALL privilege to the role `test_role` on the URI, but only the SELECT privilege on the table itself.

```
GRANT ALL ON URI 'hdfs:///user/hive/warehouse/test_db.db/test_tbl' to role test_role;
USE test_db;
GRANT SELECT ON TABLE test_tbl to role test_role;
```

With these privileges, all users with the role `test_role` will be able to carry out the `EXPORT TABLE` operation even though they should only have `SELECT` privileges on `test_db.test_tbl`:

```
EXPORT TABLE <another-table-user-can-read> TO
'hdfs:///user/hive/warehouse/test_db.db/test_tbl'
```

Debugging Failed Sentry Authorization Requests

To figure out why Sentry has denied access in a specific instance, you can temporarily enable Sentry's debug level logging:

- In Cloudera Manager, add `log4j.logger.org.apache.sentry=DEBUG` to the logging settings for your service through the corresponding **Logging Safety Valve** field for Impala or HiveServer2.
- On systems not managed by Cloudera Manager, add `log4j.logger.org.apache.sentry=DEBUG` to the `log4j.properties` file on each host in the cluster, in the appropriate configuration directory for each service.

In the logs, look for exceptions and messages such as:

```
FilePermission server..., RequestPermission server...., result [true|false]
```

which indicate each evaluation Sentry makes. The `FilePermission` is from the policy file, while `RequestPermission` is the privilege required for the query. A `RequestPermission` will iterate over all appropriate `FilePermission` settings until a match is found. If no matching privilege is found, Sentry returns `false` indicating "Access Denied" .

Symptoms of Sentry Service Failure

If the Sentry service fails and you attempt to access the Hive warehouse, Hive, Impala and HDFS will behave as follows:

- **Hive:** Queries to the Hive warehouse fail with an authentication error.
- **Impala:** The Impala Catalog server caches Sentry privileges, so if Sentry goes down, Impala queries continue to work because they are authorized against this cached copy of the metadata. However, authorization DDLs such as `CREATE ROLE` or `GRANT ROLE` will fail.
- **HDFS/Sentry Synchronized Permissions:** Affected HDFS files will continue to use a cached copy of the synchronized ACLs for a configurable period of time, after which they will fall back to NameNode ACLs. The timeout value can be modified by adding the `sentry.authorization-provider.cache-stale-threshold.ms` parameter to the `hdfs-site.xml` Safety Valve in Cloudera Manager. The default timeout value is 60 seconds, but you can increase this value from several minutes to a few hours, as needed to accommodate large clusters.

Hive SQL Syntax for Use with Sentry

Sentry permissions can be configured through Grant and Revoke statements issued either interactively or programmatically through the HiveServer2 SQL command line interface, Beeline (documentation available here). The syntax described below is very similar to the `GRANT`/`REVOKE` commands available in well-established relational database systems.

> **Important:**
>
> - When Sentry is enabled, you must use Beeline to execute Hive queries. Hive CLI is not supported with Sentry and must be disabled.
> - There are some differences in syntax between Hive and the corresponding Impala SQL statements. For the Impala syntax, see SQL Statements.

Column-level Authorization

CDH 5.5 introduces column-level access control for tables in Hive and Impala. Previously, Sentry supported privilege granularity only down to a table. Hence, if you wanted to restrict access to a column of sensitive data, the workaround would be to first create view for a subset of columns, and then grant privileges on that view. To reduce the administrative overhead associated with such an approach, Sentry now allows you to assign the `SELECT` privilege on a subset of columns in a table.

The following command grants a role the SELECT privilege on a column:

```
GRANT SELECT(column_name) ON TABLE table_name TO ROLE role_name;
```

The following command can be used to revoke the SELECT privilege on a column:

```
REVOKE SELECT(column_name) ON TABLE table_name FROM ROLE role_name;
```

Any new columns added to a table will be inaccessible by default, until explicitly granted access.

Actions allowed for users with SELECT privilege on a column:

Users whose roles have been granted the SELECT privilege on columns only, can perform operations which explicitly refer to those columns. Some examples are:

```
SELECT column_name FROM TABLE table_name;
```

In this case, Sentry will first check to see if the user has the required privileges to access the table. It will then further check to see whether the user has the SELECT privilege to access the column(s).

```
SELECT COUNT(column_name) FROM TABLE table_name;
```

Users are also allowed to use the COUNT function to return the number of values in the column.

```
SELECT column_name FROM TABLE table_name WHERE column_name <operator> GROUP BY
column_name;
```

The above command will work as long as you refer only to columns to which you already have access.

- To list the column(s) to which the current user has SELECT access:

```
SHOW COLUMNS (FROM|IN) table_name [(FROM|IN) db_name];
```

Exceptions:

- If a user has SELECT access to all columns in a table, the following command will work. Note that this is an exception, not the norm. In all other cases, SELECT on all columns does *not* allow you to perform table-level operations.

```
SELECT * FROM TABLE table_name;
```

- The DESCRIBE table command differs from the others, in that it does not filter out columns for which the user does not have SELECT access.

```
DESCRIBE (table_name);
```

Limitations:

- Column-level privileges can only be applied to tables and partitions, not views.
- **HDFS-Sentry Sync:** With HDFS-Sentry sync enabled, even if a user has been granted access to all columns of a table, they will not have access to the corresponding HDFS data files. This is because Sentry does not consider SELECT on all columns equivalent to explicitly being granted SELECT on the table.
- Column-level access control for access from Spark SQL is not supported by the HDFS-Sentry plug-in.

CREATE ROLE Statement

The CREATE ROLE statement creates a role to which privileges can be granted. Privileges can be granted to roles, which can then be assigned to users. A user that has been assigned a role will only be able to exercise the privileges of that role.

Only users that have administrative privileges can create/drop roles. By default, the `hive`, `impala` and hue users have admin privileges in Sentry.

```
CREATE ROLE [role_name];
```

DROP ROLE Statement

The `DROP ROLE` statement can be used to remove a role from the database. Once dropped, the role will be revoked for all users to whom it was previously assigned. Queries that are already executing will not be affected. However, since Hive checks user privileges before executing each query, active user sessions in which the role has already been enabled will be affected.

```
DROP ROLE [role_name];
```

GRANT ROLE Statement

The `GRANT ROLE` statement can be used to grant roles to groups. Only Sentry admin users can grant roles to a group.

```
GRANT ROLE role_name [, role_name]
    TO GROUP <groupName> [,GROUP <groupName>]
```

> **Note:** Sentry by default does not allow grants for groups with non-alphanumeric names. To work around this, use backticks around the affected group names. For example:
>
> ```
> GRANT ROLE test TO GROUP `hadoop`;
> ```

REVOKE ROLE Statement

The `REVOKE ROLE` statement can be used to revoke roles from groups. Only Sentry admin users can revoke the role from a group.

```
REVOKE ROLE role_name [, role_name]
    FROM GROUP <groupName> [,GROUP <groupName>]
```

GRANT <PRIVILEGE> Statement

To grant privileges on an object to a role, the user must be a Sentry admin user.

```
GRANT
    <PRIVILEGE> [, <PRIVILEGE> ]
    ON <OBJECT> <object_name>
    TO ROLE <roleName> [,ROLE <roleName>]
```

Starting with CDH 5.5, you can grant the `SELECT` privilege on specific columns of a table. For example:

```
GRANT SELECT(column_name) ON TABLE table_name TO ROLE role_name;
```

GRANT <PRIVILEGE> ON URIs (HDFS and S3A)

Starting with CDH 5.8, if the GRANT for Sentry URI does not specify the complete scheme, or the URI mentioned in Hive DDL statements does not have a scheme, Sentry automatically completes the URI by applying the default scheme based on the HDFS configuration provided in the `fs.defaultFS` property. Using the same HDFS configuration, Sentry can also auto-complete URIs in case the URI is missing a scheme and an authority component.

When a user attempts to access a URI, Sentry will check to see if the user has the required privileges. During the authorization check, if the URI is incomplete, Sentry will complete the URI using the default HDFS scheme. Note that Sentry does not check URI schemes for completion when they are being used to grant privileges. This is because users can GRANT privileges on URIs that do not have a complete scheme or do not already exist on the filesystem.

For example, in CDH 5.8 and higher, the following `CREATE EXTERNAL TABLE` statement works even though the statement does not include the URI scheme.

```
GRANT ALL ON URI 'hdfs://namenode:XXX/path/to/table'
CREATE EXTERNAL TABLE foo LOCATION 'namenode:XXX/path/to/table'
```

Similarly, the following `CREATE EXTERNAL TABLE` statement works even though it is missing scheme and authority components.

```
GRANT ALL ON URI 'hdfs://namenode:XXX/path/to/table'
CREATE EXTERNAL TABLE foo LOCATION '/path/to/table'
```

Since Sentry supports both HDFS and Amazon S3, starting in CDH 5.8, Cloudera recommends that you specify the fully qualified URI in GRANT statements to avoid confusion. If the underlying storage is a mix of S3 and HDFS, the risk of granting the wrong privileges increases. The following are examples of fully qualified URIs:

- **HDFS:** `hdfs://host:port/path/to/hdfs/table`
- **S3:** `s3a://host:port/path/to/s3/table`

REVOKE <PRIVILEGE> Statement

Since only authorized admin users can create roles, consequently only Sentry admin users can revoke privileges from a group.

```
REVOKE
    <PRIVILEGE> [, <PRIVILEGE> ]
    ON <OBJECT> <object_name>
    FROM ROLE <roleName> [,ROLE <roleName>]
```

You can also revoke any previously-granted `SELECT` privileges on specific columns of a table. For example:

```
REVOKE SELECT(column_name) ON TABLE table_name FROM ROLE role_name;
```

GRANT <PRIVILEGE> ... WITH GRANT OPTION

Starting with CDH 5.2, you can delegate granting and revoking privileges to other roles. For example, a role that is granted a privilege `WITH GRANT OPTION` can `GRANT/REVOKE` the same privilege to/from other roles. Hence, if a role has the ALL privilege on a database and the `WITH GRANT OPTION` set, users granted that role can execute `GRANT/REVOKE` statements only for that database or child tables of the database.

```
GRANT
    <PRIVILEGE>
    ON <OBJECT> <object_name>
    TO ROLE <roleName>
    WITH GRANT OPTION
```

Only a role with `GRANT` option on a specific privilege or its parent privilege can revoke that privilege from other roles. Once the following statement is executed, all privileges with and without grant option are revoked.

```
REVOKE
    <PRIVILEGE>
    ON <OBJECT> <object_name>
    FROM ROLE <roleName>
```

Hive does not currently support revoking only the `WITH GRANT OPTION` from a privilege previously granted to a role. To remove the `WITH GRANT OPTION`, revoke the privilege and grant it again without the `WITH GRANT OPTION` flag.

SET ROLE Statement

Sentry enforces restrictions on queries based on the roles and privileges that the user has. A user can have multiple roles and a role can have multiple privileges.

The `SET ROLE` command enforces restrictions at the role level, not at the user level. When you use the `SET ROLE` command to make a role active, the role becomes current for the session. If a role is not current for the session, it is inactive and the user does not have the privileges assigned to that role. A user can only use the `SET ROLE` command for roles that have been granted to the user.

To list the roles that are current for the user, use the `SHOW CURRENT ROLES` command. By default, all roles that are assigned to the user are current.

You can use the following `SET ROLE` commands:

SET ROLE NONE

Makes all roles for the user inactive. When no role is current, the user does not have any privileges and cannot execute a query.

SET ROLE ALL

Makes all roles that have been granted to the user active. All privileges assigned to those roles are applied. When the user executes a query, the query is filtered based on those privileges.

SET ROLE <role name>

Makes a single role active. The privileges assigned to that role are applied. When the user executes a query, the query is filtered based on the privileges assigned to that role.

SHOW Statement

- To list the database(s) for which the current user has database, table, or column-level access:

```
SHOW DATABASES;
```

- To list the table(s) for which the current user has table or column-level access:

```
SHOW TABLES;
```

- To list the column(s) to which the current user has `SELECT` access:

```
SHOW COLUMNS (FROM|IN) table_name [(FROM|IN) db_name];
```

- To list all the roles in the system (only for sentry admin users):

```
SHOW ROLES;
```

- To list all the roles in effect for the current user session:

```
SHOW CURRENT ROLES;
```

- To list all the roles assigned to the given `<groupName>` (only allowed for Sentry admin users and others users that are part of the group specified by `<groupName>`):

```
SHOW ROLE GRANT GROUP <groupName>;
```

- The `SHOW` statement can also be used to list the privileges that have been granted to a role or all the grants given to a role for a particular object.

  To list all the grants for the given `<roleName>` (only allowed for Sentry admin users and other users that have been granted the role specified by `<roleName>`). The following command will also list any column-level privileges:

```
SHOW GRANT ROLE <roleName>;
```

- To list all the grants for a role on the given `<objectName>` (only allowed for Sentry admin users and other users that have been granted the role specified by `<roleName>`). The following command will also list any column-level privileges:

```
SHOW GRANT ROLE <roleName> on <OBJECT> <objectName>;
```

Example: Using Grant/Revoke Statements to Match an Existing Policy File

> **Note:** In the following example(s), `server1` refers to an alias Sentry uses for the associated Hive service. It does not refer to any physical server. This alias can be modified using the `hive.sentry.server` property in hive-site.xml. If you are using Cloudera Manager, modify the Hive property, **Server Name for Sentry Authorization**, in the **Service-Wide** > **Advanced** category.

Here is a sample policy file:

```
[groups]
# Assigns each Hadoop group to its set of roles
manager = analyst_role, junior_analyst_role
analyst = analyst_role
jranalyst = junior_analyst_role
customers_admin = customers_admin_role
admin = admin_role

[roles] # The uris below define a define a landing skid which
# the user can use to import or export data from the system.
# Since the server runs as the user "hive" files in that directory
# must either have the group hive and read/write set or
# be world read/write.
analyst_role = server=server1->db=analyst1, \
    server=server1->db=jranalyst1->table=*->action=select
    server=server1->uri=hdfs://ha-nn-uri/landing/analyst1
junior_analyst_role = server=server1->db=jranalyst1, \
    server=server1->uri=hdfs://ha-nn-uri/landing/jranalyst1

# Implies everything on server1.
admin_role = server=server1
```

The following sections show how you can use the new GRANT statements to assign privileges to roles (and assign roles to groups) to match the sample policy file above.

Grant privileges to `analyst_role`:

```
CREATE ROLE analyst_role;
GRANT ALL ON DATABASE analyst1 TO ROLE analyst_role;
GRANT SELECT ON DATABASE jranalyst1 TO ROLE analyst_role;
GRANT ALL ON URI 'hdfs://ha-nn-uri/landing/analyst1' \
TO ROLE analyst_role;
```

Grant privileges to `junior_analyst_role`:

```
CREATE ROLE junior_analyst_role;
GRANT ALL ON DATABASE jranalyst1 TO ROLE junior_analyst_role;
GRANT ALL ON URI 'hdfs://ha-nn-uri/landing/jranalyst1' \
TO ROLE junior_analyst_role;
```

Grant privileges to `admin_role`:

```
CREATE ROLE admin_role
GRANT ALL ON SERVER server1 TO ROLE admin_role;
```

Grant roles to groups:

```
GRANT ROLE admin_role TO GROUP admin;
GRANT ROLE analyst_role TO GROUP analyst;
GRANT ROLE jranalyst_role TO GROUP jranalyst;
```

Synchronizing HDFS ACLs and Sentry Permissions

This topic introduces an HDFS-Sentry plugin that allows you to configure synchronization of Sentry privileges with HDFS ACLs for specific HDFS directories.

Previously, when Sentry was used to secure data in Hive or Impala, it was difficult to securely share the same HDFS data files with other components such as Pig, MapReduce, Spark, and HDFS clients. You had two options:

- You could set ownership for the entire Hive warehouse to `hive:hive` and not allow other components any access to the data. While this is secure, it does not allow for sharing.
- Use HDFS ACLs and synchronize Sentry privileges and HDFS ACLs manually. For example, if a user only has the Sentry SELECT privilege on a table, that user should only be able to read the table data files, and not write to those HDFS files.

Introduction

To solve the problem stated above, CDH 5.3 introduces integration of Sentry and HDFS permissions that will automatically keep HDFS ACLs in sync with the privileges configured with Sentry. This feature offers the easiest way to share data between Hive, Impala and other components such as MapReduce, Spark, and Pig, while setting permissions for that data with just one set of rules through Sentry. It maintains the ability of Hive and Impala to set permissions on views, in addition to tables, while access to data outside of Hive and Impala (for example, reading files off HDFS) requires table permissions. HDFS permissions for some or all of the files that are part of tables defined in the Hive Metastore will now be controlled by Sentry.

This change consists of three components:

- An HDFS NameNode plugin
- A Sentry-Hive Metastore plugin
- A Sentry Service plugin

With synchronization enabled, Sentry will translate permissions on tables to the appropriate corresponding HDFS ACL on the underlying table files in HDFS. For example, if a user group is assigned to a Sentry role that has SELECT permission on a particular table, then that user group will also have read access to the HDFS files that are part of that table. When you list those files in HDFS, this permission will be listed as an HDFS ACL.

Note that when Sentry was enabled, the `hive` user/group was given ownership of all files/directories in the Hive warehouse (`/user/hive/warehouse`). Hence, the resulting synchronized Sentry permissions will reflect this fact. If you skipped that step, Sentry permissions will be based on the existing Hive warehouse ACLs. Sentry will not automatically grant ownership to the`hive` user.

The mapping of Sentry privileges to HDFS ACLs is as follows:

- SELECT privilege -> Read access on the file.
- INSERT privilege -> Write access on the file.
- ALL privilege -> Read and Write access on the file.

Note that you must explicitly specify the path prefix to the Hive warehouse (default: `user/hive/warehouse`) and any other directories that must be managed by Sentry. This procedure is described in the following section.

> **Important:**
>
> - With synchronization enabled, your ability to set HDFS permissions for those files is disabled. Permissions for those particular files can be set only through Sentry, and when examined through HDFS these permissions appear as HDFS ACLs. A configurable set of users, such as `hive` and `impala`, will have full access to the files automatically. This ensures that a key requirement of using Sentry with Hive and Impala — giving these processes full access to regulate permissions on underlying data files — is met automatically.
> - Tables that are not associated with Sentry, that is, have no user with Sentry privileges to access them, will retain their old ACLs.
> - Synchronized privileges are not persisted to HDFS. This means that when this feature is disabled, HDFS privileges will return to their original values.
> - Setting HDFS ACLs on Sentry-managed paths will not affect the original HDFS ACLs. That is, if you set an ACL for a Hive object that also falls under the Sentry-managed path prefixes, no action will be taken. If the path does not point to a Hive object managed by Sentry, HDFS ACLs will be set as expected.
>
>   Removing HDFC ACLs from paths will work the same way. If you attempt to remove an ACL associated with a Hive object managed by Sentry, no action will be taken. In all other cases, the ACL will be removed as is expected behavior.
> - With HDFS-Sentry sync enabled, if the NameNode plugin is unable to communicate with the Sentry Service for a particular period of time (configurable by the `sentry.authorization-provider.cache-stale-threshold.ms` property), permissions for *all* directories under Sentry-managed path prefixes, irrespective of whether those file paths correspond to Hive warehouse objects, will be set to the Hive System User and the Hive System Group.
> - Sentry HDFS synchronization does not support Hive metastore HA.
> - Column-level access control for access from Spark SQL is not supported by the HDFS-Sentry plug-in.

Prerequisites

- CDH 5.3.0 (or higher)
- (Strongly Recommended) Implement Kerberos authentication on your cluster.

The following conditions must be also be true when enabling Sentry-HDFS synchronization. Failure to comply with any of these will result in validation errors.

- You must use the Sentry service, not policy file-based authorization.
- Enabling HDFS Extended Access Control Lists (ACLs) is required.
- There must be at least one Sentry service dependent on HDFS.
- The Sentry service must have at least one Sentry Server role.
- The Sentry service must have at least one dependent Hive service.
- The Hive service must have at least one Hive metastore role.

Enabling the HDFS-Sentry Plugin Using Cloudera Manager

1. Go to the HDFS service.
2. Click the **Configuration** tab.
3. Select **Scope** > **HDFS (Service-Wide)**.
4. Type `Check HDFS Permissions` in the Search box.
5. Select **Check HDFS Permissions**.
6. Select **Enable Sentry Synchronization**.

7. Locate the **Sentry Synchronization Path Prefixes** property or search for it by typing its name in the Search box.
8. Edit the **Sentry Synchronization Path Prefixes** property to list HDFS path prefixes where Sentry permissions should be enforced. Multiple HDFS path prefixes can be specified. By default, this property points to `/user/hive/warehouse` and must always be non-empty. If you are using a non-default location for the Hive warehouse, make sure you add it to the list of path prefixes. HDFS privilege synchronization will not occur for tables located outside the HDFS regions listed here.

> **Important:** Sentry will only manage paths that store Hive objects. If a path is listed under the **Sentry Synchronization Path Prefixes**, but there is no Hive object there, Sentry will not manage permissions for that path.

9. Click **Save Changes**.
10. Restart the cluster. Note that it may take an additional two minutes after cluster restart for privilege synchronization to take effect.

Enabling the HDFS-Sentry Plugin Using the Command Line

> **Important:**
>
> - This configuration process can be completed using either Cloudera Manager or the command-line instructions.
> - This information applies specifically to CDH 5.13.x. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

To enable the Sentry plugins on an unmanaged cluster, you must explicitly allow the `hdfs` user to interact with Sentry, and install the plugin packages as described in the following sections.

Allowing the `hdfs` user to connect with Sentry

For an unmanaged cluster, add `hdfs` to the `sentry.service.allow.connect` property in `sentry-site.xml`.

```
<property>
    <name>sentry.service.allow.connect</name>
    <value>impala,hive,hue,hdfs</value>
</property>
```

Installing the HDFS-Sentry Plugin

> **Note: Install Cloudera Repository**
>
> Before using the instructions on this page to install the package, install the Cloudera `yum`, `zypper`/YaST or `apt` repository, and install or upgrade CDH 5 and make sure it is functioning correctly. For instructions, see Installing the Latest CDH 5 Release.

Use the following the instructions, depending on your operating system, to install the `sentry-hdfs-plugin` package. The package must be installed (at a minimum) on the following hosts:

- The host running the NameNode and Secondary NameNode
- The host running the Hive Metastore
- The host running the Sentry Service

OS	Command
RHEL-compatible	`$ sudo yum install sentry-hdfs-plugin`
SLES	`$ sudo zypper install sentry-hdfs-plugin`
Ubuntu or Debian	`$ sudo apt-get install sentry-hdfs-plugin`

Configuring the HDFS NameNode Plugin

Add the following properties to the `hdfs-site.xml` file on the NameNode host.

```
<property>
<name>dfs.namenode.acls.enabled</name>
<value>true</value>
</property>

<property>
<name>dfs.namenode.authorization.provider.class</name>
<value>org.apache.sentry.hdfs.SentryAuthorizationProvider</value>
</property>

<property>
<name>dfs.permissions</name>
<value>true</value>
</property>

<!-- Comma-separated list of HDFS path prefixes where Sentry permissions should be
enforced. -->
<!-- Privilege synchronization will occur only for tables located in HDFS regions
specified here. -->
<property>
<name>sentry.authorization-provider.hdfs-path-prefixes</name>
<value>/user/hive/warehouse</value>
</property>

<property>
<name>sentry.hdfs.service.security.mode</name>
<value>kerberos</value>
</property>

<property>
<name>sentry.hdfs.service.server.principal</name>
<value> SENTRY_SERVER_PRINCIPAL (for eg :  sentry/_HOST@VPC.CLOUDERA.COM )</value>
</property>

<property>
<name>sentry.hdfs.service.client.server.rpc-port</name>
<value>SENTRY_SERVER_PORT</value>
</property>

<property>
<name>sentry.hdfs.service.client.server.rpc-address</name>
<value>SENTRY_SERVER_HOST</value>
</property>
```

Configuring the Hive Metastore Plugin

Add the following properties to `hive-site.xml` on the Hive Metastore Server host.

```
<property>
<name>sentry.metastore.plugins</name>
<value>org.apache.sentry.hdfs.MetastorePlugin</value>
</property>

<property>
<name>sentry.hdfs.service.client.server.rpc-port</name>
<value> SENTRY_SERVER_PORT </value>
</property>

<property>
<name>sentry.hdfs.service.client.server.rpc-address</name>
<value> SENTRY_SERVER_HOSTNAME </value>
</property>

<property>
<name>sentry.hdfs.service.client.server.rpc-connection-timeout</name>
<value>200000</value>
</property>
```

```
<property>
<name>sentry.hdfs.service.security.mode</name>
<value>kerberos</value>
</property>

<property>
<name>sentry.hdfs.service.server.principal</name>
<value> SENTRY_SERVER_PRINCIPAL (for eg :  sentry/_HOST@VPC.CLOUDERA.COM )</value>
</property>
```

Configuring the Sentry Service Plugin

Add the following properties to the `sentry-site.xml` file on the NameNode host.

```
<property>
<name>sentry.service.processor.factories</name>
<value>org.apache.sentry.provider.db.service.thrift.SentryPolicyStoreProcessorFactory,
org.apache.sentry.hdfs.SentryHDFSServiceProcessorFactory</value>
</property>

<property>
<name>sentry.policy.store.plugins</name>
<value>org.apache.sentry.hdfs.SentryPlugin</value>
</property>
```

> **Important:** Once all the configuration changes are complete, restart your cluster. Note that it may take an additional two minutes after cluster restart for privilege synchronization to take effect.

Testing the Sentry Synchronization Plugins

The following tasks should help you make sure that Sentry-HDFS synchronization has been enabled and configured correctly:

For a folder that has been enabled for the plugin, such as the Hive warehouse, try accessing the files in that folder outside Hive and Impala. For this, you should know what tables those HDFS files belong to and the Sentry permissions on those tables. Attempt to view or modify the Sentry permissions settings over those tables using one of the following tools:

- **(Recommended)** Hue's Security application
- HiveServer2 CLI
- Impala CLI
- Access the table files directly in HDFS. For example:
  - List files inside the folder and verify that the file permissions shown in HDFS (including ACLs) match what was configured in Sentry.
  - Run a MapReduce, Pig or Spark job that accesses those files. Pick any tool besides HiveServer2 and Impala

Using the Sentry Web Server

The Sentry webserver can be used to view reported metrics which can prove useful for debugging. To enable the Sentry webserver for reporting metrics and secure it using Kerberos authentication, perform the following steps:

1. Go to the Sentry service in Cloudera Manager.
2. Click the **Configuration** tab.
3. Select **Scope** > **Sentry (Service-Wide)**.
4. Select **Category** > **Advanced**.
5. Locate the **Sentry Service Advanced Configuration Snippet (Safety Valve) for sentry-site.xml** property and add the following properties:

a. To enable the Sentry webserver:

```
<!-- Enable the Sentry web server -->
<property>
<name>sentry.service.web.enable</name>
<value>true</value>
</property>
```

b. Metrics for the Sentry service can now be reported using either JMX or console. To obtain the metrics in JSON format, you can use the Sentry Web Server which by default, listens on port 51000. To enable reporting of metrics:

```
<!-- Port on which the Sentry web server listens -->
<property>
<name>sentry.service.web.port</name>
<value>51000</value>
</property>

<!-- Tool being used to report metrics; jmx or console -->
<property>
<name>sentry.service.reporter</name>
<value>jmx</value>
</property>
```

c. Kerberos authentication must be enabled for the Sentry web server to restrict who can access the debug webpage for the Sentry service. To enable:

```
<!-- Set Kerberos authentication properties -->
<property>
<name>sentry.service.web.authentication.type</name>
<value>KERBEROS</value>
</property>

<property>
<name>sentry.service.web.authentication.kerberos.principal</name>
<value>HTTP/<fully.qualified.domain.name>@YOUR-REALM</value>
</property>

<property>
<name>sentry.service.web.authentication.kerberos.keytab</name>
<value>/path/to/keytab/file</value>
</property>

<!-- Define comma-separated list of users allowed to connect to the web server -->
<property>
<name>sentry.service.web.authentication.allow.connect.users</name>
<value>user_a,user_b</value>
</property>
```

6. Click **Save Changes** to commit the changes.

Sentry Policy File Authorization

> **Important:** This is the documentation for configuring Sentry using the policy file approach. Cloudera recommends you use the database-backed Sentry service introduced in CDH 5.1 to secure your data. See The Sentry Service on page 351 for more information.

Sentry enables role-based, fine-grained authorization for HiveServer2, Cloudera Impala, and Cloudera Search.

The following topics provide instructions on how to install, upgrade, and configure policy file authorization.

Terminology

- An *object* is an entity protected by Sentry's authorization rules. The objects supported in the current release are `server`, `database`, `table`, `URI`, `collection`, and `config`.

- A *role* is a collection of rules for accessing a given object.
- A *privilege* is granted to a role to govern access to an object. Sentry allows you to assign the SELECT privilege to columns (only for Hive and Impala). Supported privileges are:

Table 39: Valid privilege types and the objects they apply to

Privilege	Object
INSERT	DB, TABLE
SELECT	SERVER, DB, TABLE, COLUMN
UPDATE	COLLECTION, CONFIG
QUERY	COLLECTION, CONFIG
ALL	SERVER, TABLE, DB, URI, COLLECTION, CONFIG

- A *user* is an entity that is permitted by the authentication subsystem to access the service. This entity can be a Kerberos principal, an LDAP `userid`, or an artifact of some other supported pluggable authentication system.
- A *group* connects the authentication system with the authorization system. It is a collection of one or more users who have been granted one or more authorization roles. Sentry allows a set of roles to be configured for a group.
- A configured *group provider* determines a user's affiliation with a group. The current release supports HDFS-backed groups and locally configured groups.

Privilege Model

Sentry uses a role-based privilege model with the following characteristics.

- Allows any user to execute `show function`, `desc function`, and `show locks`.
- Allows the user to see only those tables, databases, collections, and configurations for which the user has privileges.
- Requires a user to have the necessary privileges on the URI to execute HiveQL operations that specify a location. Examples of such operations include `LOAD`, `IMPORT`, and `EXPORT`.
- Privileges granted on URIs are recursively applied to all subdirectories. That is, privileges only need to be granted on the parent directory.
- Sentry provides column-level access control for tables in Hive and Impala. You can assign the `SELECT` privilege on a subset of columns in a table.

Important:

- When Sentry is enabled, you must use Beeline to execute Hive queries. Hive CLI is not supported with Sentry and must be disabled.
- When Sentry is enabled, a user with no privileges on a database will not be allowed to connect to HiveServer2. This is because the `use <database>` command is executed as part of the connection to HiveServer2, which is why the connection fails. See HIVE-4256.

For more information, see Authorization Privilege Model for Hive and Impala on page 400 and Authorization Privilege Model for Solr on page 403.

Granting Privileges

Note: In the following example(s), `server1` refers to an alias Sentry uses for the associated Hive service. It does not refer to any physical server. This alias can be modified using the `hive.sentry.server` property in hive-site.xml. If you are using Cloudera Manager, modify the Hive property, **Server Name for Sentry Authorization**, in the **Service-Wide** > **Advanced** category.

For example, a rule for the `Select` privilege on table `customers` from database `sales` is formulated as follows:

```
server=server1->db=sales->table=customer->action=Select
```

To assign the `Select` privilege to the `sales_read` role on the `Id` column from the `customers` table, the rule would be as follows:

```
sales_read = server=server1->db=sales->table=customers->column=Id->action=select
```

Each object must be specified as a hierarchy of the containing objects, from server to table, followed by the privilege granted for that object. A role can contain multiple such rules, separated by commas. For example, a role might contain the `Select` privilege for the `customer` and `items` tables in the `sales` database, and the `Insert` privilege for the `sales_insights` table in the `reports` database. You would specify this as follows:

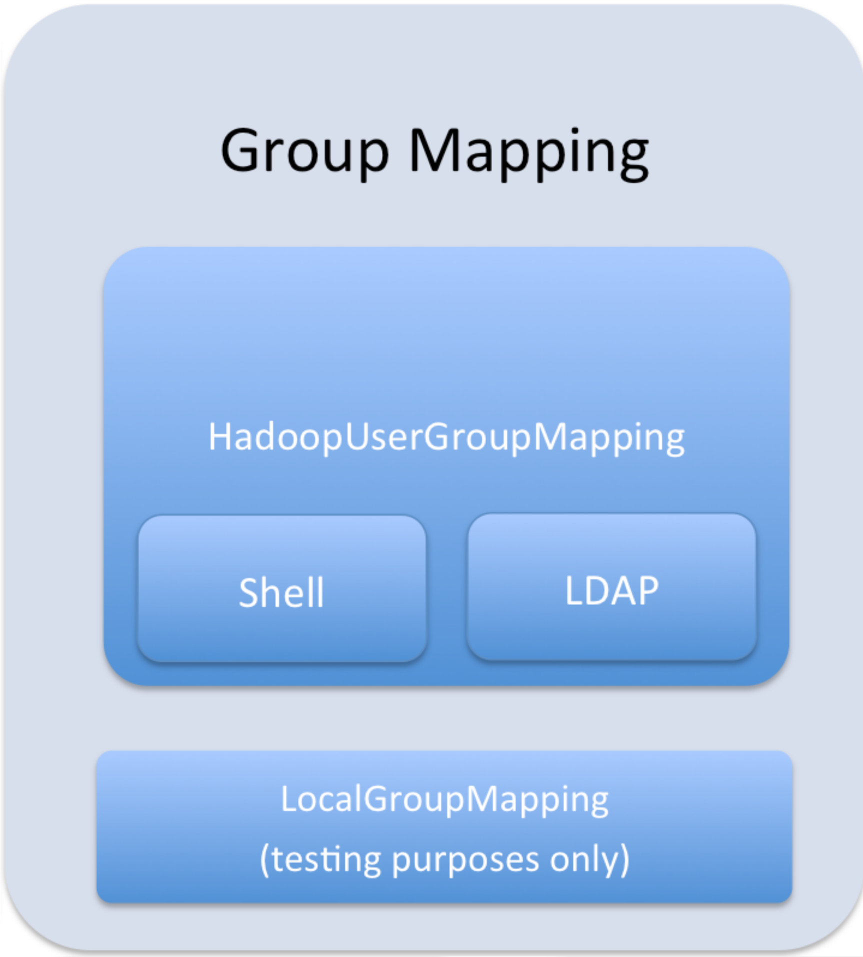
```
sales_reporting =                                            \
server=server1->db=sales->table=customer->action=Select,     \
server=server1->db=sales->table=items->action=Select,         \
server=server1->db=reports->table=sales_insights->action=Insert
```

User to Group Mapping

You can configure Sentry to use either Hadoop groups or groups defined in the policy file. By default, Sentry looks up groups locally, but it can be configured to look up Hadoop groups using LDAP (for Active Directory). User/group information for Sentry, Hive and Impala must be made available for lookup on the following hosts:

- Sentry - Groups are looked up on the host the Sentry Server runs on.
- Hive - Groups are looked up on the hosts running HiveServer2 and the Hive Metastore.
- Impala - Groups are looked up on the Catalog Server and on all of the Impala daemon hosts.

Group mappings in Sentry can be summarized as in the figure below:



Important: You can use either Hadoop groups or local groups, but not both at the same time. Local groups are traditionally used for a quick proof-of-concept, while Hadoop groups are more commonly used in production. Refer Configuring LDAP Group Mappings on page 343 for details on configuring LDAP group mappings in Hadoop.

Policy File

The sections that follow contain notes on creating and maintaining the policy file, and using URIs to load external data and JARs.

Warning: An invalid policy file will be ignored while logging an exception. This will lead to a situation where users will lose access to all Sentry-protected data, since default Sentry behavior is *deny* unless a user has been explicitly granted access. (Note that if only the per-DB policy file is invalid, it will invalidate only the policies in that file.)

Storing the Policy File

Considerations for storing the policy file(s) in HDFS include:

1. Replication count - Because the file is read for each query in Hive and read once every five minutes by all Impala daemons, you should increase this value; since it is a small file, setting the replication count equal to the number of client nodes in the cluster is reasonable.

2. Updating the file - Updates to the file are reflected immediately, so you should write them to a temporary copy of the file first, and then replace the existing file with the temporary one after all the updates are complete. This avoids race conditions caused by reads on an incomplete file.

Defining Roles

Keep in mind that role definitions are not cumulative; the definition that is further down in the file replaces the older one. For example, the following results in `role1` having `privilege2`, not `privilege1` and `privilege2`.

```
role1 = privilege1
role1 = privilege2
```

Role names are scoped to a specific file. For example, if you give `role1` the `ALL` privilege on `db1` in the global policy file and give `role1` `ALL` on `db2` in the per-db db2 policy file, the user will be given both privileges.

URIs

Any command which references a URI such as `CREATE TABLE EXTERNAL`, `LOAD`, `IMPORT`, `EXPORT`, and more, in addition to `CREATE TEMPORARY FUNCTION` requires the `URI` privilege. This is an important security control because without this users could simply create an external table over an existing table they do not have access to and bypass Sentry.

URIs must start with either `hdfs://` or `file://`. If a URI starts with anything else, it will cause an exception and the policy file will be invalid.

When defining URIs for HDFS, you must also specify the NameNode. For example:

```
data_read = server=server1->uri=file:///path/to/dir,\
server=server1->uri=hdfs://namenode:port/path/to/dir
```

Important: Because the NameNode host and port must be specified, Cloudera strongly recommends you use High Availability (HA). This ensures that the URI will remain constant even if the NameNode changes.

Loading Data

Data can be loaded using a landing skid, either in HDFS or using a local/NFS directory where HiveServer2/Impala run. The following privileges can be used to grant a role access to a loading skid:

- **Load data from a local/NFS directory:**

```
server=server1->uri=file:///path/to/nfs/local/to/nfs
```

- **Load data from HDFS (MapReduce, Pig, and so on):**

```
server=server1->uri=hdfs://ha-nn-uri/data/landing-skid
```

In addition to the privilege in Sentry, the `hive` or `impala` user will require the appropriate file permissions to access the data being loaded. Groups can be used for this purpose. For example, create a group `hive-users`, and add the `hive` and `impala` users along with the users who will be loading data, to this group.

The example `usermod` and `groupadd` commands below are only applicable to locally defined groups on the NameNode, JobTracker, and ResourceManager. If you use another system for group management, equivalent changes should be made in your group management system.

```
$ groupadd hive-users
$ usermod -G someuser,hive-users someuser
$ usermod -G hive,hive-users hive
```

External Tables

External tables require the `ALL@database` privilege in addition to the `URI` privilege. When data is being inserted through the `EXTERNAL TABLE` statement, or is referenced from an HDFS location outside the normal Hive database directories, the user needs appropriate permissions on the URIs corresponding to those HDFS locations. This means that the URI location must either be owned by the `hive:hive` user OR the `hive/impala` users must be members of the group that owns the directory.

You can configure access to the directory using a URI as follows:

```
[roles]
someuser_home_dir_role = server=server1->uri=hdfs://ha-nn-uri/user/someuser
```

You should now be able to create an external table:

```
CREATE EXTERNAL TABLE ...
LOCATION 'hdfs://ha-nn-uri/user/someuser/mytable';
```

Sample Sentry Configuration Files

This section provides a sample configuration.

> **Note:** In the following example(s), `server1` refers to an alias Sentry uses for the associated Hive service. It does not refer to any physical server. This alias can be modified using the `hive.sentry.server` property in hive-site.xml. If you are using Cloudera Manager, modify the Hive property, **Server Name for Sentry Authorization**, in the **Service-Wide** > **Advanced** category.

Policy Files

The following is an example of a policy file with a per-DB policy file. In this example, the first policy file, `sentry-provider.ini` would exist in HDFS; `hdfs://ha-nn-uri/etc/sentry/sentry-provider.ini` might be an appropriate location. The per-DB policy file is for the customer's database. It is located at `hdfs://ha-nn-uri/etc/sentry/customers.ini`.

sentry-provider.ini

```
[databases]
# Defines the location of the per DB policy file for the customers DB/schema
customers = hdfs://ha-nn-uri/etc/sentry/customers.ini

[groups]
# Assigns each Hadoop group to its set of roles
manager = analyst_role, junior_analyst_role
analyst = analyst_role
jranalyst = junior_analyst_role
customers_admin = customers_admin_role
admin = admin_role

[roles]
# The uris below define a define a landing skid which
# the user can use to import or export data from the system.
# Since the server runs as the user "hive" files in that directory
# must either have the group hive and read/write set or
# be world read/write.
analyst_role = server=server1->db=analyst1, \
    server=server1->db=jranalyst1->table=*->action=select
    server=server1->uri=hdfs://ha-nn-uri/landing/analyst1
junior_analyst_role = server=server1->db=jranalyst1, \
    server=server1->uri=hdfs://ha-nn-uri/landing/jranalyst1

# Implies everything on server1 -> customers. Privileges for
# customers can be defined in the global policy file even though
# customers has its only policy file. Note that the Privileges from
# both the global policy file and the per-DB policy file
# are merged. There is no overriding.
```

```
customers_admin_role = server=server1->db=customers

# Implies everything on server1.
admin_role = server=server1
```

customers.ini

```
[groups]
manager = customers_insert_role, customers_select_role
analyst = customers_select_role

[roles]
customers_insert_role = server=server1->db=customers->table=*->action=insert
customers_select_role = server=server1->db=customers->table=*->action=select
```

> **Important:** Sentry does not support using the `view` keyword in policy files. If you want to define a role against a view, use the keyword `table` instead. For example, to define the role `analyst_role` against the view `col_test_view`:
>
> ```
> [roles]
> analyst_role =
> server=server1->db=default->table=col_test_view->action=select
> ```

Sentry Configuration File

The following is an example of a `sentry-site.xml` file.

> **Important:** If you are using Cloudera Manager 4.6 (or lower), make sure you **do not** store `sentry-site.xml` in `/etc/hive/conf` ; that directory is regenerated whenever the Hive client configurations are redeployed. Instead, use a directory such as `/etc/sentry` to store the `sentry` file.
>
> If you are using Cloudera Manager 4.7 (or higher), Cloudera Manager will create and deploy `sentry-site.xml` for you.See The Sentry Service on page 351 for more details on configuring Sentry with Cloudera Manager.

sentry-site.xml

```
<configuration>
  <property>
    <name>hive.sentry.provider</name>

<value>org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider</value>

  </property>

  <property>
    <name>hive.sentry.provider.resource</name>
    <value>/path/to/authz-provider.ini</value>
    <!--
       If the hdfs-site.xml points to HDFS, the path will be in HDFS;
       alternatively you could specify a full path, e.g.:
       hdfs://namenode:port/path/to/authz-provider.ini
       file:///path/to/authz-provider.ini
    -->
  </property>

  <property>
    <name>sentry.hive.server</name>
    <value>server1</value>
  </property>
</configuration>
```

Accessing Sentry-Secured Data Outside Hive/Impala

When Sentry is enabled, the `hive` user owns all data within the Hive warehouse. However, unlike traditional database systems the enterprise data hub allows for multiple engines to execute over the same dataset.

> **Note:** Cloudera strongly recommends you use Hive/Impala SQL queries to access data secured by Sentry, as opposed to accessing the data files directly.

However, there are scenarios where fully vetted and reviewed jobs will also need to access the data stored in the Hive warehouse. A typical scenario would be a secured MapReduce transformation job that is executed automatically as an application user. In such cases it's important to know that the user executing this job will also have full access to the data in the Hive warehouse.

Scenario One: Authorizing Jobs

Problem

A reviewed, vetted, and automated job requires access to the Hive warehouse and cannot use Hive/Impala to access the data.

Solution

Create a group which contains `hive`, `impala`, and the user executing the automated job. For example, if the `etl` user is executing the automated job, you can create a group called `hive-users` which contains the `hive`, `impala`, and `etl` users.

The example `usermod` and groupadd commands below are only applicable to locally defined groups on the NameNode, JobTracker, and ResourceManager. If you use another system for group management, equivalent changes should be made in your group management system.

```
$ groupadd hive-users
$ usermod -G hive,impala,hive-users hive
$ usermod -G hive,impala,hive-users impala
$ usermod -G etl,hive-users etl
```

Once you have added users to the `hive-users` group, change directory permissions in the HDFS:

```
$ hadoop fs -chgrp -R hive:hive-users /user/hive/warehouse
$ hadoop fs -chmod -R 770 /user/hive/warehouse
```

Scenario Two: Authorizing Group Access to Databases

Problem

One group of users, `grp1` should have full access to the database, `db1`, outside of Sentry. The database, `db1` should not be accessible to any other groups, outside of Sentry. Sentry should be used for all other authorization needs.

Solution

Place the `hive` and `impala` users in `grp1`.

```
$ usermod -G hive,impala,grp1 hive
$ usermod -G hive,impala,grp1 impala
```

Then change group ownerships of all directories and files in `db1` to `grp1`, and modify directory permissions in the HDFS. This example is only applicable to local groups on a single host.

```
$ hadoop fs -chgrp -R hive:grp1 /user/hive/warehouse/db1.db
$ hadoop fs -chmod -R 770 /user/hive/warehouse/db1.db
```

Debugging Failed Sentry Authorization Requests

Sentry logs all facts that lead up to authorization decisions at the debug level. If you do not understand why Sentry is denying access, the best way to debug is to temporarily turn on debug logging:

- In Cloudera Manager, add `log4j.logger.org.apache.sentry=DEBUG` to the logging settings for your service through the corresponding **Logging Safety Valve** field for the Impala, Hive Server 2, or Solr Server services.
- On systems not managed by Cloudera Manager, add `log4j.logger.org.apache.sentry=DEBUG` to the `log4j.properties` file on each host in the cluster, in the appropriate configuration directory for each service.

Specifically, look for exceptions and messages such as:

```
FilePermission server..., RequestPermission server...., result [true|false]
```

which indicate each evaluation Sentry makes. The `FilePermission` is from the policy file, while `RequestPermission` is the privilege required for the query. A `RequestPermission` will iterate over all appropriate `FilePermission` settings until a match is found. If no matching privilege is found, Sentry returns `false` indicating "Access Denied" .

Authorization Privilege Model for Hive and Impala

Privileges can be granted on different objects in the Hive warehouse. Any privilege that can be granted is associated with a level in the object hierarchy. If a privilege is granted on a container object in the hierarchy, the base object automatically inherits it. For instance, if a user has `ALL` privileges on the database scope, then (s)he has `ALL` privileges on all of the base objects contained within that scope. Sentry 5.13 and above supports Hive metastore high availability.

Object Hierarchy

```
Server
     URI
     Database
         Table
             Partition
             Columns
         View
```

Table 40: Valid privilege types and objects they apply to

Privilege	Object
INSERT	DB, TABLE
SELECT	DB, TABLE, VIEW, COLUMN
ALL	SERVER, TABLE, DB, URI

Table 41: Privilege hierarchy

Base Object	Granular privileges on object	Container object that contains the base object	Privileges on container object that implies privileges on the base object
DATABASE	ALL	SERVER	ALL
TABLE	INSERT	DATABASE	ALL
TABLE	SELECT	DATABASE	ALL
COLUMN	SELECT	DATABASE	ALL
VIEW	SELECT	DATABASE	ALL

Table 42: Privilege table for Hive & Impala operations

Operation	Scope	Privileges Required	URI
CREATE DATABASE	SERVER	ALL	

Operation	Scope	Privileges Required	URI
DROP DATABASE	DATABASE	ALL	
CREATE TABLE	DATABASE	ALL	
DROP TABLE	TABLE	ALL	
CREATE VIEW -This operation is allowed if you have column-level SELECT access to the columns being used.	DATABASE; SELECT on TABLE;	ALL	
ALTER VIEW -This operation is allowed if you have column-level SELECT access to the columns being used.	VIEW/TABLE	ALL	
DROP VIEW	VIEW/TABLE	ALL	
ALTER TABLE .. ADD COLUMNS	TABLE	ALL	
ALTER TABLE .. REPLACE COLUMNS	TABLE	ALL	
ALTER TABLE .. CHANGE column	TABLE	ALL	
ALTER TABLE .. RENAME	TABLE	ALL	
ALTER TABLE .. SET TBLPROPERTIES	TABLE	ALL	
ALTER TABLE .. SET FILEFORMAT	TABLE	ALL	
ALTER TABLE .. SET LOCATION	TABLE	ALL	URI
ALTER TABLE .. ADD PARTITION	TABLE	ALL	
ALTER TABLE .. ADD PARTITION location	TABLE	ALL	URI
ALTER TABLE .. DROP PARTITION	TABLE	ALL	
ALTER TABLE .. PARTITION SET FILEFORMAT	TABLE	ALL	
SHOW CREATE TABLE	TABLE	SELECT/INSERT	
SHOW PARTITIONS	TABLE	SELECT/INSERT	
SHOW TABLES -Output includes all the tables for which the user has table-level privileges and all the tables for which the user has some column-level privileges.	TABLE	SELECT/INSERT	
SHOW GRANT ROLE -Output includes an additional field for any column-level privileges.	TABLE	SELECT/INSERT	
DESCRIBE TABLE	TABLE	SELECT/INSERT	

Operation	Scope	Privileges Required	URI
-Output shows *all* columns if the user has table level-privileges or SELECT privilege on at least one table column			
LOAD DATA	TABLE	INSERT	URI
SELECT -You can grant the SELECT privilege on a view to give users access to specific columns of a table they do not otherwise have access to. -See Column-level Authorization on page 381 for details on allowed column-level operations.	VIEW/TABLE; COLUMN	SELECT	
INSERT OVERWRITE TABLE	TABLE	INSERT	
CREATE TABLE .. AS SELECT -This operation is allowed if you have column-level SELECT access to the columns being used.	DATABASE; SELECT on TABLE	ALL	
USE <dbName>	Any		
CREATE FUNCTION	SERVER	ALL	
ALTER TABLE .. SET SERDEPROPERTIES	TABLE	ALL	
ALTER TABLE .. PARTITION SET SERDEPROPERTIES	TABLE	ALL	
Hive-Only Operations			
INSERT OVERWRITE DIRECTORY	TABLE	INSERT	URI
Analyze TABLE	TABLE	SELECT + INSERT	
IMPORT TABLE	DATABASE	ALL	URI
EXPORT TABLE	TABLE	SELECT	URI
ALTER TABLE TOUCH	TABLE	ALL	
ALTER TABLE TOUCH PARTITION	TABLE	ALL	
ALTER TABLE .. CLUSTERED BY SORTED BY	TABLE	ALL	
ALTER TABLE .. ENABLE/DISABLE	TABLE	ALL	
ALTER TABLE .. PARTITION ENABLE/DISABLE	TABLE	ALL	
ALTER TABLE .. PARTITION.. RENAME TO PARTITION	TABLE	ALL	
MSCK REPAIR TABLE	TABLE	ALL	
ALTER DATABASE	DATABASE	ALL	

Operation	Scope	Privileges Required	URI
DESCRIBE DATABASE	DATABASE	SELECT/INSERT	
SHOW COLUMNS -Output for this operation filters columns to which the user does not have explicit SELECT access	TABLE	SELECT/INSERT	
CREATE INDEX	TABLE	ALL	
DROP INDEX	TABLE	ALL	
SHOW INDEXES	TABLE	SELECT/INSERT	
GRANT PRIVILEGE	Allowed only for Sentry admin users		
REVOKE PRIVILEGE	Allowed only for Sentry admin users		
SHOW GRANT	Allowed only for Sentry admin users		
SHOW TBLPROPERTIES	TABLE	SELECT/INSERT	
DESCRIBE TABLE .. PARTITION	TABLE	SELECT/INSERT	
ADD JAR	Not Allowed		
ADD FILE	Not Allowed		
DFS	Not Allowed		
Impala-Only Operations			
EXPLAIN	TABLE; COLUMN	SELECT	
INVALIDATE METADATA	SERVER	ALL	
INVALIDATE METADATA <table name>	TABLE	SELECT/INSERT	
REFRESH <table name> or REFRESH <table name> PARTITION (<partition_spec>)	TABLE	SELECT/INSERT	
DROP FUNCTION	SERVER	ALL	
COMPUTE STATS	TABLE	ALL	

Authorization Privilege Model for Solr

The tables below refer to the request handlers defined in the generated `solrconfig.xml.secure`. If you are not using this configuration file, the below may not apply.

`admin` is a special collection in sentry used to represent administrative actions. A non-administrative request may only require privileges on the collection or config on which the request is being performed. This is called either `collection1` or `config1` in this appendix. An administrative request may require privileges on both the `admin` collection and `collection1`. This is denoted as `admin, collection1` in the tables below.

> **Note:** If no privileges are granted, no access is possible. For example, accessing the Solr Admin UI requires the QUERY privilege. If no users are granted the QUERY privilege, no access to the Solr Admin UI is possible.

Table 43: Privilege table for non-administrative request handlers

Request Handler	Required Collection Privilege	Collections that Require Privilege
select	QUERY	collection1
query	QUERY	collection1
get	QUERY	collection1
browse	QUERY	collection1
tvrh	QUERY	collection1
clustering	QUERY	collection1
terms	QUERY	collection1
elevate	QUERY	collection1
analysis/field	QUERY	collection1
analysis/document	QUERY	collection1
update	UPDATE	collection1
update/json	UPDATE	collection1
update/csv	UPDATE	collection1

Table 44: Privilege table for collections admin actions

Collection Action	Required Collection Privilege	Collections that Require Privilege
create	UPDATE	admin, collection1
delete	UPDATE	admin, collection1
reload	UPDATE	admin, collection1
createAlias	UPDATE	admin, collection1 **Note:** collection1 here refers to the name of the alias, not the underlying collection(s). For example, http://YOUR-HOST:8983/solr/admin/collections?action=CREATEALIAS&name=collection1&collections=underlyingCollection
deleteAlias	UPDATE	admin, collection1 **Note:** collection1 here refers to the name of the alias, not the underlying collection(s). For example, http://YOUR-HOST:8983/solr/admin/collections?action=DELETEALIAS&name=collection1

Collection Action	Required Collection Privilege	Collections that Require Privilege
syncShard	UPDATE	admin, collection1
splitShard	UPDATE	admin, collection1
deleteShard	UPDATE	admin, collection1

Table 45: Privilege table for core admin actions

Collection Action	Required Collection Privilege	Collections that Require Privilege
create	UPDATE	admin, collection1
rename	UPDATE	admin, collection1
load	UPDATE	admin, collection1
unload	UPDATE	admin, collection1
status	UPDATE	admin, collection1
persist	UPDATE	admin
reload	UPDATE	admin, collection1
swap	UPDATE	admin, collection1
mergeIndexes	UPDATE	admin, collection1
split	UPDATE	admin, collection1
prepRecover	UPDATE	admin, collection1
requestRecover	UPDATE	admin, collection1
requestSyncShard	UPDATE	admin, collection1
requestApplyUpdates	UPDATE	admin, collection1

Table 46: Privilege table for Info and AdminHandlers

Request Handler	Required Collection Privilege	Collections that Require Privilege
LukeRequestHandler	QUERY	admin
SystemInfoHandler	QUERY	admin
SolrInfoMBeanHandler	QUERY	admin
PluginInfoHandler	QUERY	admin
ThreadDumpHandler	QUERY	admin
PropertiesRequestHandler	QUERY	admin
LogginHandler	QUERY, UPDATE (or *)	admin
ShowFileRequestHandler	QUERY	admin

Table 47: Privilege table for Config Admin actions

Config Action	Required Collection Privilege	Collections that Require Privilege	Required Config Privilege	Configs that Require Privilege
CREATE	UPDATE	admin	*	config1

Config Action	Required Collection Privilege	Collections that Require Privilege	Required Config Privilege	Configs that Require Privilege
DELETE	UPDATE	admin	*	config1

Installing and Upgrading Sentry for Policy File Authorization

Sentry stores the configuration as well as privilege policies in files. The `sentry-site.xml` file contains configuration options such as `group association provider`, `privilege policy file location`. The policy file contains the privileges and groups. It has a `.ini` file format and can be stored on a local file system or HDFS.

Sentry is plugged into Hive as session hooks, which you configure in `hive-site.xml`. The `sentry` package must be installed. It contains the required JAR files. You must also configure properties in the Sentry Configuration File on page 398.

> **Important:**
>
> Install the Cloudera `yum`, `zypper/YaST` or `apt` repository before using the following commands. For instructions, see Installing the Latest CDH 5 Release.

Installing Sentry

Use the following the instructions, depending on your operating system, to install the latest version of Sentry.

> **Important: Configuration files**
>
> - If you install a newer version of a package that is already on the system, configuration files that you have modified will remain intact.
> - If you uninstall a package, the package manager renames any configuration files you have modified from `<file>` to `<file>.rpmsave`. If you then re-install the package (probably to install a new version) the package manager creates a new `<file>` with applicable defaults. You are responsible for applying any changes captured in the original configuration file to the new configuration file. In the case of Ubuntu and Debian upgrades, you will be prompted if you have made changes to a file for which there is a new version. For details, see Automatic handling of configuration files by `dpkg`.

OS	Command
RHEL	`$ sudo yum install sentry`
SLES	`$ sudo zypper install sentry`
Ubuntu or Debian	`$ sudo apt-get update;` `$ sudo apt-get install sentry`

Upgrading Sentry

If you are upgrading from CDH 5.x to the latest CDH release, see Installing Sentry on page 406 to install the latest version.

Configuring Sentry Policy File Authorization Using Cloudera Manager

This topic describes how to configure Sentry policy files and enable policy file authorization for CDH services using Cloudera Manager.

Configuring User to Group Mappings

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

Hadoop Groups

1. Go to the Hive service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Hive (Service-Wide)**.
4. Select **Category** > **Policy File Based Sentry**.
5. Locate the **Sentry User to Group Mapping Class** property or search for it by typing its name in the Search box.
6. Set the **Sentry User to Group Mapping Class** property to `org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider`.
7. Click **Save Changes**.
8. Restart the Hive service.

Local Groups

> **Note:** You can use either Hadoop groups or local groups, but not both at the same time. Use local groups if you want to do a quick proof-of-concept. For production, use Hadoop groups.

1. Define local groups in the `[users]` section of the Policy File on page 395. For example:

```
[users]
user1 = group1, group2, group3
user2 = group2, group3
```

2. Modify Sentry configuration as follows:
   a. Go to the Hive service.
   b. Click the **Configuration** tab.
   c. Select **Scope** > **Hive (Service-Wide)**.
   d. Select **Category** > **Policy File Based Sentry**.
   e. Locate the **Sentry User to Group Mapping Class** property or search for it by typing its name in the Search box.
   f. Set the **Sentry User to Group Mapping Class** property to `org.apache.sentry.provider.file.LocalGroupResourceAuthorizationProvider`.
   g. Click **Save Changes**.
   h. Restart the Hive service.

Enabling URIs for Per-DB Policy Files

The `ADD JAR` command does *not* work with HiveServer2 and the Beeline client when Beeline runs on a different host. As an alternative to `ADD JAR`, Hive's *auxiliary paths* functionality should be used as described in the following steps.

> **Important:** Enabling URIs in per-DB policy files introduces a security risk by allowing the owner of the db-level policy file to grant himself/herself load privileges to anything the `hive` user has read permissions for in HDFS (including data in other databases controlled by different db-level policy files).

Add the following string to the Java configuration options for HiveServer2 during startup.

```
-Dsentry.allow.uri.db.policyfile=true
```

Using User-Defined Functions with HiveServer2

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

The `ADD JAR` command does *not* work with HiveServer2 and the Beeline client when Beeline runs on a different host. As an alternative to `ADD JAR`, Hive's *auxiliary paths* functionality should be used. There are some differences in the procedures for creating permanent functions and temporary functions. For detailed instructions, see Using Cloudera Manager to Create User-Defined Functions (UDFs) with HiveServer2.

Enabling Policy File Authorization for Hive

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

1. See Before You Install Sentry on page 359 to verify the prerequisites.
2. Setting Hive Warehouse Directory Permissions

> **Important:** If you are going to enable HDFS/Sentry synchronization, you do not need to perform the following step to explicitly set permissions for the Hive warehouse directory. With synchronization enabled, all Hive databases and tables will automatically be owned by `hive:hive`, and Sentry permissions on tables are translated to HDFS ACLs for the underlying table files.

The Hive warehouse directory (`/user/hive/warehouse` or any path you specify as `hive.metastore.warehouse.dir` in your `hive-site.xml`) must be owned by the Hive user and group.

- **Using the default Hive warehouse directory** - Permissions on the warehouse directory must be set as follows (see following Note for caveats):
  - **771** on the directory itself (by default, `/user/hive/warehouse`)
  - **771** on all subdirectories (for example, `/user/hive/warehouse/mysubdir`)
  - All files and subdirectories should be owned by hive:hive

  For example:

```
$ sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

  If you have enabled Kerberos on your cluster, you must kinit as the `hdfs` user before you set permissions. For example:

```
sudo -u hdfs kinit -kt <hdfs.keytab> hdfs
sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

- **Using a non-default Hive warehouse:** If you would like to use a different directory as the Hive warehouse, update the `hive.metastore.warehouse.dir` property, and make sure you set the required permissions on the new directory. For example, if the new warehouse directory is `/data`, set the permissions as follows:

```
$ hdfs dfs -chown hive:hive /data
$ hdfs dfs -chmod 771 /data
```

  Note that when you update the default Hive warehouse, previously created tables will not be moved over automatically. Therefore, tables created before the update will remain at `/user/hive/warehouse/<old_table>`. However, after the update, any new tables created in the default location will be found at `/data/<new_table>`.

  For Sentry/HDFS sync to work as expected, add the new warehouse URL to the list of **Sentry Synchronization Path Prefixes**.

> **Note:**
>
> - If you set `hive.warehouse.subdir.inherit.perms` to `true` in `hive-site.xml`, the permissions on the subdirectories will be set when you set permissions on the warehouse directory.
> - If a user has access to any object in the warehouse, that user will be able to execute `use default`. This ensures that `use default` commands issued by legacy applications work when Sentry is enabled.
> - The instructions described above for modifying permissions on the Hive warehouse directory override the recommendations in the Hive section of the CDH 5 Installation Guide.

3. Disable impersonation for HiveServer2:
   a. Go to the Hive service.
   b. Click the **Configuration** tab.
   c. Select **Scope** > **HiveServer2**.
   d. Select **Category** > **All**.
   e. Locate the **HiveServer2 Enable Impersonation** property or search for it by typing its name in the Search box.
   f. Under the **HiveServer2** role group, clear the **HiveServer2 Enable Impersonation** property.
   g. Click **Save Changes** to commit the changes.
4. Create the Sentry policy file, `sentry-provider.ini`, as an HDFS file.
5. Enable the Hive user to submit MapReduce jobs.
   a. Go to the MapReduce service.
   b. Click the **Configuration** tab.
   c. Select **Scope** > **TaskTracker**.
   d. Select **Category** > **Security**.
   e. Locate the **Minimum User ID for Job Submission** property or search for it by typing its name in the Search box.
   f. Set the **Minimum User ID for Job Submission** property to `0` (the default is `1000`).
   g. Click **Save Changes** to commit the changes.
   h. Repeat steps 5.a-5.d for *every* TaskTracker role group for the MapReduce service that is associated with Hive, if more than one exists.
   i. Restart the MapReduce service.
6. Enable the Hive user to submit YARN jobs.
   a. Go to the YARN service.
   b. Click the **Configuration** tab.
   c. Select **Scope** > **NodeManager**.
   d. Select **Category** > **Security**.
   e. Ensure the **Allowed System Users** property includes the `hive` user. If not, add `hive`.
   f. Click **Save Changes** to commit the changes.
   g. Repeat steps 6.a-6.d for *every* NodeManager role group for the YARN service that is associated with Hive, if more than one exists.
   h. Restart the YARN service.
7. Go to the Hive service.
8. Click the **Configuration** tab.
9. Select **Scope** > **Hive (Service-Wide)**.
10. Select **Category** > **Policy File Based Sentry**.
11. Select **Enable Sentry Authorization Using Policy Files**.
12. Click **Save Changes** to commit the changes.

13. Add the Hive user group to Sentry's admin groups.

    a. Go to the Sentry service.
    b. Click the **Configuration** tab.
    c. Select **Scope** > **Sentry (Service-Wide)**.
    d. Select **Category** > **Main**.
    e. Locate the **Admin Groups** property and add the `hive` group to the list. If an end user is in one of these admin groups, that user has administrative privileges on the Sentry Server.
    f. Click **Save Changes** to commit the changes.

14. Restart the cluster and HiveServer2 after changing these values, whether you use Cloudera Manager or not.

Configuring Group Access to the Hive Metastore

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

You can configure the Hive Metastore to reject connections from users not listed in the Hive group proxy list (in HDFS). If you do not configure this override, the Hive Metastore will use the value in the core-site HDFS configuration. To configure the Hive group proxy list:

1. Go to the Hive service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Hive (Service-Wide)**.
4. Select **Category** > **Proxy**.
5. In the **Hive Metastore Access Control and Proxy User Groups Override** property, specify a list of groups whose users are allowed to access the Hive Metastore. If you do not specify "*" (wildcard), you will be warned if the groups do not include hive and impala (if the Impala service is configured) in the list of groups.
6. Click **Save Changes**.
7. Restart the Hive service.

Enabling Policy File Authorization for Impala

For a cluster managed by Cloudera Manager, perform the following steps to enable policy file authorization for Impala.

1. Enable Sentry's policy file based authorization for Hive. For details, see Enabling Policy File Authorization for Hive on page 408.
2. Go to the Cloudera Manager Admin Console and go to the Impala service.
3. Click the **Configuration** tab.
4. Select **Scope** > **Impala (Service-Wide)**.
5. Select **Category** > **Policy File-Based Sentry**.
6. Select **Enable Sentry Authorization Using Policy Files**.
7. Click **Save Changes** to commit the changes.
8. Restart the Impala service.
9. Add the Impala user group to Sentry's admin groups.

    a. Go to the Sentry service.
    b. Click the **Configuration** tab.
    c. Select **Scope** > **Sentry (Service-Wide)**.
    d. Select **Category** > **Main**.
    e. Locate the **Admin Groups** property and add the `impala` group to the list. If an end user is in one of these admin groups, that user has administrative privileges on the Sentry Server.
    f. Click **Save Changes** to commit the changes.

For more details, see Starting the impalad Daemon with Sentry Authorization Enabled on page 416.

Enabling Sentry Policy File Authorization for Solr

Minimum Required Role: Full Administrator

1. Ensure the following requirements are satisfied:
   - Cloudera Search 1.1.1 or higher or CDH 5 or higher.
   - A secure Hadoop cluster.
2. Create the policy file `sentry-provider.ini` as an HDFS file. When you create the policy file `sentry-provider.ini` follow the instructions in the Policy File section in Solr Authentication on page 151. The file must be owned by owned by the `solr` user in the `solr` group, with perms=600. By default Cloudera Manager assumes the policy file is in the HDFS location `/user/solr/sentry`. To configure the location:
   a. Go to the Solr service.
   b. Click the **Configuration** tab.
   c. Select **Scope** > **SOLR (Service-Wide)**.
   d. Select **Category** > **Policy File Based Sentry**.
   e. Locate the **Sentry Global Policy File** property.
   f. Modify the path in the **Sentry Global Policy File** property.
   g. Select **Enable Sentry Authorization**.
   h. Click **Save Changes**.
3. Restart the Solr service.

For more details, see Configuring Sentry Authorization for Cloudera Search on page 425.

Configuring Sentry to Enable BDR Replication

Cloudera recommends the following steps when configuring Sentry and data replication is enabled.

- Group membership should be managed outside of Sentry (as OS and LDAP groups are typically managed) and replication for them also should be handled outside of Cloudera Manager.
- In Cloudera Manager, set up HDFS replication for the Sentry files of the databases that are being replicated (separately using Hive/Impala replication).
- On the source cluster:
  - Use a separate Sentry policy file for every database
  - Avoid placing any group or role info (except for server admin info) in the global Sentry policy file (to avoid manual replication/merging with the global file on the target cluster)
  - To avoid manual fix up of URI privileges, ensure that the URIs for the data are the same on both the source and target cluster
- On the target cluster:
  - In the global Sentry policy file, manually add the *DB name - DB file* mapping entries for the databases being replicated
  - Manually copy the server admin info from the global Sentry policy file on the source to the policy on the target cluster
  - For the databases being replicated, avoid adding more privileges (adding tables specific to target cluster may sometimes require adding extra privileges to allow access to those tables). If any target cluster specific privileges absolutely need to be added for a database, add them to the global Sentry policy file on the target cluster since the per database files would be overwritten periodically with source versions during scheduled replication.

Configuring Sentry Policy File Authorization Using the Command Line

This topic describes how to configure Sentry policy files and enable policy file authorization for unmanaged CDH services using the command line.

Configuring User to Group Mappings

Hadoop Groups

Set the `hive.sentry.provider` property in `sentry-site.xml`.

```
<property>
<name>hive.sentry.provider</name>
<value>org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider</value>
</property>
```

Local Groups

1. Define local groups in the `[users]` section of the Policy File on page 395. For example:

```
[users]
user1 = group1, group2, group3
user2 = group2, group3
```

2. Modify Sentry configuration as follows:

   In `sentry-site.xml`, set `hive.sentry.provider` as follows:

```
<property>
<name>hive.sentry.provider</name>
<value>org.apache.sentry.provider.file.LocalGroupResourceAuthorizationProvider</value>
</property>
```

Enabling URIs for Per-DB Policy Files

The ADD JAR command does *not* work with HiveServer2 and the Beeline client when Beeline runs on a different host. As an alternative to ADD JAR, Hive's *auxiliary paths* functionality should be used as described in the following steps.

> **Important:** Enabling URIs in per-DB policy files introduces a security risk by allowing the owner of the db-level policy file to grant himself/herself load privileges to anything the `hive` user has read permissions for in HDFS (including data in other databases controlled by different db-level policy files).

Add the following string to the Java configuration options for HiveServer2 during startup.

```
-Dsentry.allow.uri.db.policyfile=true
```

Using User-Defined Functions with HiveServer2

The ADD JAR command does not work with HiveServer2 and the Beeline client when Beeline runs on a different host. As an alternative to ADD JAR, Hive's *auxiliary paths* functionality should be used as described in the following steps. There are some differences in the procedures for creating permanent functions and temporary functions. For detailed instructions, see User-Defined Functions (UDFs) with HiveServer2 Using the Command Line.

Enabling Policy File Authorization for Hive

Prerequisites

- See Before You Install Sentry on page 359 to verify the prerequisites for Sentry.
- Setting Hive Warehouse Directory Permissions

> **Important:** If you are going to enable HDFS/Sentry synchronization, you do not need to perform the following step to explicitly set permissions for the Hive warehouse directory. With synchronization enabled, all Hive databases and tables will automatically be owned by `hive:hive`, and Sentry permissions on tables are translated to HDFS ACLs for the underlying table files.

The Hive warehouse directory (`/user/hive/warehouse` or any path you specify as `hive.metastore.warehouse.dir` in your `hive-site.xml`) must be owned by the Hive user and group.

- **Using the default Hive warehouse directory** - Permissions on the warehouse directory must be set as follows (see following Note for caveats):
  - **771** on the directory itself (by default, `/user/hive/warehouse`)
  - **771** on all subdirectories (for example, `/user/hive/warehouse/mysubdir`)
  - All files and subdirectories should be owned by hive:hive

  For example:

```
$ sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

  If you have enabled Kerberos on your cluster, you must kinit as the `hdfs` user before you set permissions. For example:

```
sudo -u hdfs kinit -kt <hdfs.keytab> hdfs
sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

- **Using a non-default Hive warehouse:** If you would like to use a different directory as the Hive warehouse, update the `hive.metastore.warehouse.dir` property, and make sure you set the required permissions on the new directory. For example, if the new warehouse directory is `/data`, set the permissions as follows:

```
$ hdfs dfs -chown hive:hive /data
$ hdfs dfs -chmod 771 /data
```

  Note that when you update the default Hive warehouse, previously created tables will not be moved over automatically. Therefore, tables created before the update will remain at `/user/hive/warehouse/<old_table>`. However, after the update, any new tables created in the default location will be found at `/data/<new_table>`.

  For Sentry/HDFS sync to work as expected, add the new warehouse URL to the list of **Sentry Synchronization Path Prefixes**.

> **Note:**
>
> - If you set `hive.warehouse.subdir.inherit.perms` to `true` in `hive-site.xml`, the permissions on the subdirectories will be set when you set permissions on the warehouse directory.
> - If a user has access to any object in the warehouse, that user will be able to execute `use default`. This ensures that `use default` commands issued by legacy applications work when Sentry is enabled.
> - The instructions described above for modifying permissions on the Hive warehouse directory override the recommendations in the Hive section of the CDH 5 Installation Guide.

- HiveServer2 impersonation must be turned off.
- The Hive user must be able to submit MapReduce jobs. You can ensure that this is true by setting the minimum user ID for job submission to 0. Edit the `taskcontroller.cfg` file and set `min.user.id=0`.

  To enable the Hive user to submit YARN jobs, add the user `hive` to the `allowed.system.users` configuration property. Edit the `container-executor.cfg` file and add `hive` to the `allowed.system.users` property. For example,

```
allowed.system.users = nobody,impala,hive,hbase
```

> **Important:**
> - You must restart the cluster and HiveServer2 after changing this value, whether you use Cloudera Manager or not.
> - These instructions override the instructions under Configuring MRv1 Security on page 102
> - These instructions override the instructions under Configuring YARN Security on page 104

- Add the Hive, Impala, and Hue groups to Sentry's admin groups. If an end user is in one of these admin groups, that user has administrative privileges on the Sentry Server.

```
<property>
  <name>sentry.service.admin.group</name>
  <value>hive,impala,hue</value>
 </property>
```

Configuration Changes Required

To enable Sentry, add the following properties to `hive-site.xml`:

```
<property>
<name>hive.server2.session.hook</name>
<value>org.apache.sentry.binding.hive.HiveAuthzBindingSessionHook</value>
</property>

<property>
<name>hive.sentry.conf.url</name>
<value></value>
<description>sentry-site.xml file location</description>
</property>

<property>
<name>hive.metastore.client.impl</name>
<value>org.apache.sentry.binding.metastore.SentryHiveMetaStoreClient</value>
<description>Sets custom Hive Metastore client which Sentry uses to filter out
metadata.</description>
</property>
```

Securing the Hive Metastore

It's important that the Hive metastore be secured. If you want to override the Kerberos prerequisite for the Hive metastore, set the `sentry.hive.testing.mode` property to `true` to allow Sentry to work with weaker authentication mechanisms. Add the following property to the HiveServer2 and Hive metastore's `sentry-site.xml`:

```
<property>
  <name>sentry.hive.testing.mode</name>
  <value>true</value>
</property>
```

Impala does not require this flag to be set.

> **Warning:** Cloudera strongly recommends against enabling this property in production. Use Sentry's testing mode only in test environments.

You can turn on Hive metastore security using the instructions in Cloudera Security. To secure the Hive metastore; see Hive Metastore Server Security Configuration on page 132.

Enabling Policy File Authorization for Impala

First, enable Sentry's policy file based authorization for Hive. For details, see Enabling Policy File Authorization for Hive on page 412.

See Enabling Sentry Authorization for Impala on page 415 for details on configuring Impala to work with Sentry policy files.

Enabling Sentry in Cloudera Search

See Enabling Solr as a Client for the Sentry Service Using the Command Line on page 372 for details on enabling Sentry for Solr.

See Using Solr with the Sentry Service on page 428 for details on securing Solr data.

Enabling Sentry Authorization for Impala

Authorization determines which users are allowed to access which resources, and what operations they are allowed to perform. In Impala 1.1 and higher, you use the Sentry open source project for authorization. Sentry adds a fine-grained authorization framework for Hadoop. By default (when authorization is not enabled), Impala does all read and write operations with the privileges of the `impala` user, which is suitable for a development/test environment but not for a secure production environment. When authorization is enabled, Impala uses the OS user ID of the user who runs `impala-shell` or other client program, and associates various privileges with each user.

> **Note:** Sentry is typically used in conjunction with Kerberos authentication, which defines which hosts are allowed to connect to each server. Using the combination of Sentry and Kerberos prevents malicious users from being able to connect by creating a named account on an untrusted machine. See Enabling Kerberos Authentication for Impala on page 141 for details about Kerberos authentication.

The Sentry Privilege Model

Privileges can be granted on different objects in the schema. Any privilege that can be granted is associated with a level in the object hierarchy. If a privilege is granted on a container object in the hierarchy, the child object automatically inherits it. This is the same privilege model as Hive and other database systems such as MySQL.

The object hierarchy for Impala covers Server, URI, Database, Table, and Column. (The Table privileges apply to views as well; anywhere you specify a table name, you can specify a view name instead.) Column-level authorization is available in and higher, as described in Column-level Authorization on page 381. Previously, you constructed views to query specific columns and assigned privileges based on the views rather than the base tables.

A restricted set of privileges determines what you can do with each object:

SELECT privilege

Lets you read data from a table or view, for example with the `SELECT` statement, the `INSERT...SELECT` syntax, or `CREATE TABLE...LIKE`. Also required to issue the `DESCRIBE` statement or the `EXPLAIN` statement for a query against a particular table. Only objects for which a user has this privilege are shown in the output for `SHOW DATABASES` and `SHOW TABLES` statements. The `REFRESH` statement and `INVALIDATE METADATA` statements only access metadata for tables for which the user has this privilege.

INSERT privilege

Lets you write data to a table. Applies to the `INSERT` and `LOAD DATA` statements.

ALL privilege

Lets you create or modify the object. Required to run DDL statements such as `CREATE TABLE`, `ALTER TABLE`, or `DROP TABLE` for a table, `CREATE DATABASE` or `DROP DATABASE` for a database, or `CREATE VIEW`, `ALTER VIEW`, or `DROP VIEW` for a view. Also required for the URI of the “location” parameter for the `CREATE EXTERNAL TABLE` and `LOAD DATA` statements.

Privileges can be specified for a table or view before that object actually exists. If you do not have sufficient privilege to perform an operation, the error message does not disclose if the object exists or not.

Originally, privileges were encoded in a policy file, stored in HDFS. This mode of operation is still an option, but the emphasis of privilege management is moving towards being SQL-based. Although currently Impala does not have `GRANT` or `REVOKE` statements, Impala can make use of privileges assigned through `GRANT` and `REVOKE` statements done through Hive. The mode of operation with `GRANT` and `REVOKE` statements instead of the policy file requires that a special Sentry service be enabled; this service stores, retrieves, and manipulates privilege information stored inside the metastore database.

Starting the impalad Daemon with Sentry Authorization Enabled

To run the `impalad` daemon with authorization enabled, you add one or more options to the `IMPALA_SERVER_ARGS` declaration in the `/etc/default/impala` configuration file:

- The `-server_name` option turns on Sentry authorization for Impala. The authorization rules refer to a symbolic server name, and you specify the name to use as the argument to the `-server_name` option.
- If you specify just `-server_name`, Impala uses the Sentry service for authorization, relying on the results of `GRANT` and `REVOKE` statements issued through Hive. (This mode of operation is available in Impala 1.4.0 and higher.) Prior to Impala 1.4.0, or if you want to continue storing privilege rules in the policy file, also specify the `-authorization_policy_file` option as in the following item.
- Specifying the `-authorization_policy_file` option in addition to `-server_name` makes Impala read privilege information from a policy file, rather than from the metastore database. The argument to the `-authorization_policy_file` option specifies the HDFS path to the policy file that defines the privileges on different schema objects.

For example, you might adapt your `/etc/default/impala` configuration to contain lines like the following. To use the Sentry service rather than the policy file:

```
IMPALA_SERVER_ARGS=" \
-server_name=server1 \
...
```

Or to use the policy file, as in releases prior to Impala 1.4:

```
IMPALA_SERVER_ARGS=" \
-authorization_policy_file=/user/hive/warehouse/auth-policy.ini \
-server_name=server1 \
...
```

The preceding examples set up a symbolic name of `server1` to refer to the current instance of Impala. This symbolic name is used in the following ways:

- In an environment managed by Cloudera Manager, the server name is specified through **Impala (Service-Wide)** > **Category** > **Advanced** > **Sentry Service** and **Hive** > **Service-Wide** > **Advanced** > **Sentry Service**. The values must be the same for both, so that Impala and Hive can share the privilege rules. Restart the Impala and Hive services after setting or changing this value.

- In an environment not managed by Cloudera Manager, you specify this value for the `sentry.hive.server` property in the `sentry-site.xml` configuration file for Hive, as well as in the `-server_name` option for `impalad`.

  If the `impalad` daemon is not already running, start it as described in [Starting Impala](). If it is already running, restart it with the command `sudo /etc/init.d/impala-server restart`. Run the appropriate commands on all the nodes where `impalad` normally runs.

- If you use the mode of operation using the policy file, the rules in the `[roles]` section of the policy file refer to this same `server1` name. For example, the following rule sets up a role `report_generator` that lets users with that role query any table in a database named `reporting_db` on a node where the `impalad` daemon was started up with the `-server_name=server1` option:

```
[roles]
report_generator = server=server1->db=reporting_db->table=*->action=SELECT
```

When `impalad` is started with one or both of the `-server_name=server1` and `-authorization_policy_file` options, Impala authorization is enabled. If Impala detects any errors or inconsistencies in the authorization settings or the policy file, the daemon refuses to start.

Using Impala with the Sentry Service (or higher only)

When you use the Sentry service rather than the policy file, you set up privileges through GRANT and REVOKE statement in either Impala or Hive, then both components use those same privileges automatically. (Impala added the GRANT and REVOKE statements in .)

Hive already had GRANT and REVOKE statements prior to CDH 5.1, but those statements were not production-ready. CDH 5.1 is the first release where those statements use the Sentry framework and are considered GA level. If you used the Hive GRANT and REVOKE statements prior to CDH 5.1, you must set up these privileges with the CDH 5.1 versions of GRANT and REVOKE to take advantage of Sentry authorization.

For information about using the updated Hive GRANT and REVOKE statements, see Sentry service topic in the *CDH 5 Security Guide*.

Using Impala with the Sentry Policy File

The policy file is a file that you put in a designated location in HDFS, and is read during the startup of the `impalad` daemon when you specify both the `-server_name` and `-authorization_policy_file` startup options. It controls which objects (databases, tables, and HDFS directory paths) can be accessed by the user who connects to `impalad`, and what operations that user can perform on the objects.

Note:

In CDH 5 and higher, Cloudera recommends managing privileges through SQL statements, as described in Using Impala with the Sentry Service (or higher only) on page 417. If you are still using policy files, plan to migrate to the new approach some time in the future.

The location of the policy file is listed in the `auth-site.xml` configuration file. To minimize overhead, the security information from this file is cached by each `impalad` daemon and refreshed automatically, with a default interval of 5 minutes. After making a substantial change to security policies, restart all Impala daemons to pick up the changes immediately.

Policy File Location and Format

The policy file uses the familiar `.ini` format, divided into the major sections `[groups]` and `[roles]`. There is also an optional `[databases]` section, which allows you to specify a specific policy file for a particular database, as explained in Using Multiple Policy Files for Different Databases on page 421. Another optional section, `[users]`, allows you to override the OS-level mapping of users to groups; that is an advanced technique primarily for testing and debugging, and is beyond the scope of this document.

In the `[groups]` section, you define various categories of users and select which roles are associated with each category. The group and usernames correspond to Linux groups and users on the server where the `impalad` daemon runs.

The group and usernames in the `[groups]` section correspond to Linux groups and users on the server where the `impalad` daemon runs. When you access Impala through the `impalad` interpreter, for purposes of authorization, the user is the logged-in Linux user and the groups are the Linux groups that user is a member of. When you access Impala through the ODBC or JDBC interfaces, the user and password specified through the connection string are used as login credentials for the Linux server, and authorization is based on that username and the associated Linux group membership.

In the `[roles]` section, you a set of roles. For each role, you specify precisely the set of privileges is available. That is, which objects users with that role can access, and what operations they can perform on those objects. This is the lowest-level category of security information; the other sections in the policy file map the privileges to higher-level divisions of groups and users. In the `[groups]` section, you specify which roles are associated with which groups. The group and usernames correspond to Linux groups and users on the server where the `impalad` daemon runs. The privileges are specified using patterns like:

```
server=server_name->db=database_name->table=table_name->action=SELECT
server=server_name->db=database_name->table=table_name->action=CREATE
server=server_name->db=database_name->table=table_name->action=ALL
```

For the *server_name* value, substitute the same symbolic name you specify with the `impalad -server_name` option. You can use `*` wildcard characters at each level of the privilege specification to allow access to all such objects. For example:

```
server=impala-host.example.com->db=default->table=t1->action=SELECT
server=impala-host.example.com->db=*->table=*->action=CREATE
server=impala-host.example.com->db=*->table=audit_log->action=SELECT
server=impala-host.example.com->db=default->table=t1->action=*
```

When authorization is enabled, Impala uses the policy file as a *whitelist*, representing every privilege available to any user on any object. That is, only operations specified for the appropriate combination of object, role, group, and user are allowed; all other operations are not allowed. If a group or role is defined multiple times in the policy file, the last definition takes precedence.

To understand the notion of whitelisting, set up a minimal policy file that does not provide any privileges for any object. When you connect to an Impala node where this policy file is in effect, you get no results for `SHOW DATABASES`, and an error when you issue any `SHOW TABLES`, `USE database_name`, `DESCRIBE table_name`, `SELECT`, and or other statements that expect to access databases or tables, even if the corresponding databases and tables exist.

The contents of the policy file are cached, to avoid a performance penalty for each query. The policy file is re-checked by each `impalad` node every 5 minutes. When you make a non-time-sensitive change such as adding new privileges or new users, you can let the change take effect automatically a few minutes later. If you remove or reduce privileges, and want the change to take effect immediately, restart the `impalad` daemon on all nodes, again specifying the `-server_name` and `-authorization_policy_file` options so that the rules from the updated policy file are applied.

Examples of Policy File Rules for Security Scenarios

The following examples show rules that might go in the policy file to deal with various authorization-related scenarios. For illustration purposes, this section shows several very small policy files with only a few rules each. In your environment, typically you would define many roles to cover all the scenarios involving your own databases, tables, and applications, and a smaller number of groups, whose members are given the privileges from one or more roles.

A User with No Privileges

If a user has no privileges at all, that user cannot access any schema objects in the system. The error messages do not disclose the names or existence of objects that the user is not authorized to read.

This is the experience you want a user to have if they somehow log into a system where they are not an authorized Impala user. In a real deployment with a filled-in policy file, a user might have no privileges because they are not a member of any of the relevant groups mentioned in the policy file.

Examples of Privileges for Administrative Users

When an administrative user has broad access to tables or databases, the associated rules in the `[roles]` section typically use wildcards and/or inheritance. For example, in the following sample policy file, `db=*` refers to all databases and `db=*->table=*` refers to all tables in all databases.

Omitting the rightmost portion of a rule means that the privileges apply to all the objects that could be specified there. For example, in the following sample policy file, the `all_databases` role has all privileges for all tables in all databases, while the `one_database` role has all privileges for all tables in one specific database. The `all_databases` role does not grant privileges on URIs, so a group with that role could not issue a `CREATE TABLE` statement with a `LOCATION` clause. The `entire_server` role has all privileges on both databases and URIs within the server.

```
[groups]
supergroup = all_databases

[roles]
read_all_tables = server=server1->db=*->table=*->action=SELECT
all_tables = server=server1->db=*->table=*
all_databases = server=server1->db=*
one_database = server=server1->db=test_db
entire_server = server=server1
```

A User with Privileges for Specific Databases and Tables

If a user has privileges for specific tables in specific databases, the user can access those things but nothing else. They can see the tables and their parent databases in the output of `SHOW TABLES` and `SHOW DATABASES`, `USE` the appropriate databases, and perform the relevant actions (`SELECT` and/or `INSERT`) based on the table privileges. To actually create a table requires the `ALL` privilege at the database level, so you might define separate roles for the user that sets up a schema and other users or applications that perform day-to-day operations on the tables.

The following sample policy file shows some of the syntax that is appropriate as the policy file grows, such as the # comment syntax, \ continuation syntax, and comma separation for roles assigned to groups or privileges assigned to roles.

```
[groups]
cloudera = training_sysadmin, instructor
visitor = student

[roles]
training_sysadmin = server=server1->db=training, \
server=server1->db=instructor_private, \
server=server1->db=lesson_development
instructor = server=server1->db=training->table=*->action=*, \
server=server1->db=instructor_private->table=*->action=*, \
server=server1->db=lesson_development->table=lesson*
# This particular course is all about queries, so the students can SELECT but not INSERT
 or CREATE/DROP.
student = server=server1->db=training->table=lesson_*->action=SELECT
```

Privileges for Working with External Data Files

When data is being inserted through the `LOAD DATA` statement, or is referenced from an HDFS location outside the normal Impala database directories, the user also needs appropriate permissions on the URIs corresponding to those HDFS locations.

In this sample policy file:

- The `external_table` role lets us insert into and query the Impala table, `external_table.sample`.
- The `staging_dir` role lets us specify the HDFS path `/user/cloudera/external_data` with the `LOAD DATA` statement. Remember, when Impala queries or loads data files, it operates on all the files in that directory, not just a single file, so any Impala `LOCATION` parameters refer to a directory rather than an individual file.
- We included the IP address and port of the Hadoop name node in the HDFS URI of the `staging_dir` rule. We found those details in `/etc/hadoop/conf/core-site.xml`, under the `fs.default.name` element. That is what we use in any roles that specify URIs (that is, the locations of directories in HDFS).
- We start this example after the table `external_table.sample` is already created. In the policy file for the example, we have already taken away the `external_table_admin` role from the `cloudera` group, and replaced it with the lesser-privileged `external_table` role.
- We assign privileges to a subdirectory underneath `/user/cloudera` in HDFS, because such privileges also apply to any subdirectories underneath. If we had assigned privileges to the parent directory `/user/cloudera`, it would be too likely to mess up other files by specifying a wrong location by mistake.
- The `cloudera` under the `[groups]` section refers to the `cloudera` group. (In the demo VM used for this example, there is a `cloudera` user that is a member of a `cloudera` group.)

Policy file:

```
[groups]
cloudera = external_table, staging_dir

[roles]
external_table_admin = server=server1->db=external_table
external_table = server=server1->db=external_table->table=sample->action=*
staging_dir =
server=server1->uri=hdfs://127.0.0.1:8020/user/cloudera/external_data->action=*
```

`impala-shell` session:

```
[localhost:21000] > use external_table;
Query: use external_table
[localhost:21000] > show tables;
Query: show tables
Query finished, fetching results ...
+--------+
| name   |
+--------+
| sample |
+--------+
Returned 1 row(s) in 0.02s

[localhost:21000] > select * from sample;
Query: select * from sample
Query finished, fetching results ...
+-----+
| x   |
+-----+
| 1   |
| 5   |
| 150 |
+-----+
Returned 3 row(s) in 1.04s

[localhost:21000] > load data inpath '/user/cloudera/external_data' into table sample;
Query: load data inpath '/user/cloudera/external_data' into table sample
Query finished, fetching results ...
+----------------------------------------------------------+
| summary                                                  |
+----------------------------------------------------------+
| Loaded 1 file(s). Total files in destination location: 2 |
+----------------------------------------------------------+
Returned 1 row(s) in 0.26s
[localhost:21000] > select * from sample;
Query: select * from sample
Query finished, fetching results ...
+-------+
| x     |
+-------+
| 2     |
| 4     |
| 6     |
| 8     |
| 64738 |
| 49152 |
| 1     |
| 5     |
| 150   |
+-------+
Returned 9 row(s) in 0.22s

[localhost:21000] > load data inpath '/user/cloudera/unauthorized_data' into table
sample;
Query: load data inpath '/user/cloudera/unauthorized_data' into table sample
ERROR: AuthorizationException: User 'cloudera' does not have privileges to access:
hdfs://127.0.0.1:8020/user/cloudera/unauthorized_data
```

Separating Administrator Responsibility from Read and Write Privileges

Remember that to create a database requires full privilege on that database, while day-to-day operations on tables within that database can be performed with lower levels of privilege on specific table. Thus, you might set up separate roles for each database or application: an administrative one that could create or drop the database, and a user-level one that can access only the relevant tables.

For example, this policy file divides responsibilities between users in 3 different groups:

- Members of the `supergroup` group have the `training_sysadmin` role and so can set up a database named `training`.

- Members of the `cloudera` group have the `instructor` role and so can create, insert into, and query any tables in the `training` database, but cannot create or drop the database itself.
- Members of the `visitor` group have the `student` role and so can query those tables in the `training` database.

```
[groups]
supergroup = training_sysadmin
cloudera = instructor
visitor = student

[roles]
training_sysadmin = server=server1->db=training
instructor = server=server1->db=training->table=*->action=*
student = server=server1->db=training->table=*->action=SELECT
```

Using Multiple Policy Files for Different Databases

For an Impala cluster with many databases being accessed by many users and applications, it might be cumbersome to update the security policy file for each privilege change or each new database, table, or view. You can allow security to be managed separately for individual databases, by setting up a separate policy file for each database:

- Add the optional `[databases]` section to the main policy file.
- Add entries in the `[databases]` section for each database that has its own policy file.
- For each listed database, specify the HDFS path of the appropriate policy file.

For example:

```
[databases]
# Defines the location of the per-DB policy files for the 'customers' and 'sales'
databases.
customers = hdfs://ha-nn-uri/etc/access/customers.ini
sales = hdfs://ha-nn-uri/etc/access/sales.ini
```

To enable URIs in per-DB policy files, add the following string in the Cloudera Manager field **Impala Service Environment Advanced Configuration Snippet (Safety Valve)**:

```
JAVA_TOOL_OPTIONS="-Dsentry.allow.uri.db.policyfile=true"
```

> **Important:** Enabling URIs in per-DB policy files introduces a security risk by allowing the owner of the db-level policy file to grant himself/herself load privileges to anything the `impala` user has read permissions for in HDFS (including data in other databases controlled by different db-level policy files).

Setting Up Schema Objects for a Secure Impala Deployment

Remember that in your role definitions, you specify privileges at the level of individual databases and tables, or all databases or all tables within a database. To simplify the structure of these rules, plan ahead of time how to name your schema objects so that data with different authorization requirements is divided into separate databases.

If you are adding security on top of an existing Impala deployment, remember that you can rename tables or even move them between databases using the `ALTER TABLE` statement. In Impala, creating new databases is a relatively inexpensive operation, basically just creating a new directory in HDFS.

You can also plan the security scheme and set up the policy file before the actual schema objects named in the policy file exist. Because the authorization capability is based on whitelisting, a user can only create a new database or table if the required privilege is already in the policy file: either by listing the exact name of the object being created, or a * wildcard to match all the applicable objects within the appropriate container.

Privilege Model and Object Hierarchy

Privileges can be granted on different objects in the schema. Any privilege that can be granted is associated with a level in the object hierarchy. If a privilege is granted on a container object in the hierarchy, the child object automatically inherits it. This is the same privilege model as Hive and other database systems such as MySQL.

The kinds of objects in the schema hierarchy are:

```
Server
    URI
    Database
        Table
```

The server name is specified by the `-server_name` option when `impalad` starts. Specify the same name for all `impalad` nodes in the cluster.

URIs represent the HDFS paths you specify as part of statements such as `CREATE EXTERNAL TABLE` and `LOAD DATA`. Typically, you specify what look like UNIX paths, but these locations can also be prefixed with `hdfs://` to make clear that they are really URIs. To set privileges for a URI, specify the name of a directory, and the privilege applies to all the files in that directory and any directories underneath it.

In and higher, you can specify privileges for individual columns, as described in Column-level Authorization on page 381. Formerly, to specify read privileges at this level, you created a view that queried specific columns and/or partitions from a base table, and gave `SELECT` privilege on the view but not the underlying table.

URIs must start with either `hdfs://` or `file://`. If a URI starts with anything else, it will cause an exception and the policy file will be invalid. When defining URIs for HDFS, you must also specify the NameNode. For example:

```
data_read = server=server1->uri=file:///path/to/dir, \
server=server1->uri=hdfs://namenode:port/path/to/dir
```

Warning:

Because the NameNode host and port must be specified, Cloudera strongly recommends you use High Availability (HA). This ensures that the URI will remain constant even if the NameNode changes.

```
data_read = server=server1->uri=file:///path/to/dir,\
server=server1->uri=hdfs://ha-nn-uri/path/to/dir
```

Table 48: Valid privilege types and objects they apply to

Privilege	Object
INSERT	DB, TABLE
SELECT	DB, TABLE, COLUMN
ALL	SERVER, TABLE, DB, URI

Note:

Although this document refers to the `ALL` privilege, currently if you use the policy file mode, you do not use the actual keyword `ALL` in the policy file. When you code role entries in the policy file:

- To specify the `ALL` privilege for a server, use a role like `server=`*`server_name`*.
- To specify the `ALL` privilege for a database, use a role like `server=`*`server_name`*`->db=`*`database_name`*.
- To specify the `ALL` privilege for a table, use a role like `server=`*`server_name`*`->db=`*`database_name`*`->table=`*`table_name`*`->action=*`.

Operation	Scope	Privileges	URI
EXPLAIN	TABLE; COLUMN	SELECT	
LOAD DATA	TABLE	INSERT	URI

Operation	Scope	Privileges	URI
CREATE DATABASE	SERVER	ALL	
DROP DATABASE	DATABASE	ALL	
CREATE TABLE	DATABASE	ALL	
DROP TABLE	TABLE	ALL	
DESCRIBE TABLE -Output shows *all* columns if the user has table level-privileges or SELECT privilege on at least one table column	TABLE	SELECT/INSERT	
ALTER TABLE .. ADD COLUMNS	TABLE	ALL on DATABASE	
ALTER TABLE .. REPLACE COLUMNS	TABLE	ALL on DATABASE	
ALTER TABLE .. CHANGE column	TABLE	ALL on DATABASE	
ALTER TABLE .. RENAME	TABLE	ALL on DATABASE	
ALTER TABLE .. SET TBLPROPERTIES	TABLE	ALL on DATABASE	
ALTER TABLE .. SET FILEFORMAT	TABLE	ALL on DATABASE	
ALTER TABLE .. SET LOCATION	TABLE	ALL on DATABASE	URI
ALTER TABLE .. ADD PARTITION	TABLE	ALL on DATABASE	
ALTER TABLE .. ADD PARTITION location	TABLE	ALL on DATABASE	URI
ALTER TABLE .. DROP PARTITION	TABLE	ALL on DATABASE	
ALTER TABLE .. PARTITION SET FILEFORMAT	TABLE	ALL on DATABASE	
ALTER TABLE .. SET SERDEPROPERTIES	TABLE	ALL on DATABASE	
CREATE VIEW -This operation is allowed if you have column-level SELECT access to the columns being used.	DATABASE; SELECT on TABLE;	ALL	
DROP VIEW	VIEW/TABLE	ALL	
ALTER VIEW	You need ALL privilege on the named view and the parent database, plus SELECT privilege for any tables or views referenced by the view query. Once the view is created or altered by a high-privileged system administrator, it can be queried by a lower-privileged user who does not have full	ALL, SELECT	

Operation	Scope	Privileges	URI
	query privileges for the base tables.		
ALTER TABLE .. SET LOCATION	TABLE	ALL on DATABASE	URI
CREATE EXTERNAL TABLE	Database (ALL), URI (SELECT)	ALL, SELECT	
SELECT -You can grant the SELECT privilege on a view to give users access to specific columns of a table they do not otherwise have access to. -See Column-level Authorization on page 381 for details on allowed column-level operations.	VIEW/TABLE; COLUMN	SELECT	
USE <dbName>	Any		
CREATE FUNCTION	SERVER	ALL	
DROP FUNCTION	SERVER	ALL	
REFRESH <table name> or REFRESH <table name> PARTITION (<partition_spec>)	TABLE	SELECT/INSERT	
INVALIDATE METADATA	SERVER	ALL	
INVALIDATE METADATA <table name>	TABLE	SELECT/INSERT	
COMPUTE STATS	TABLE	ALL	
SHOW TABLE STATS, SHOW PARTITIONS	TABLE	SELECT/INSERT	
SHOW COLUMN STATS	TABLE	SELECT/INSERT	
SHOW FUNCTIONS	DATABASE	SELECT	
SHOW TABLES		No special privileges needed to issue the statement, but only shows objects you are authorized for	
SHOW DATABASES, SHOW SCHEMAS		No special privileges needed to issue the statement, but only shows objects you are authorized for	

Debugging Failed Sentry Authorization Requests

Sentry logs all facts that lead up to authorization decisions at the debug level. If you do not understand why Sentry is denying access, the best way to debug is to temporarily turn on debug logging:

- In Cloudera Manager, add `log4j.logger.org.apache.sentry=DEBUG` to the logging settings for your service through the corresponding **Logging Safety Valve** field for the Impala, Hive Server 2, or Solr Server services.

- On systems not managed by Cloudera Manager, add `log4j.logger.org.apache.sentry=DEBUG` to the `log4j.properties` file on each host in the cluster, in the appropriate configuration directory for each service.

Specifically, look for exceptions and messages such as:

```
FilePermission server..., RequestPermission server...., result [true|false]
```

which indicate each evaluation Sentry makes. The `FilePermission` is from the policy file, while `RequestPermission` is the privilege required for the query. A `RequestPermission` will iterate over all appropriate `FilePermission` settings until a match is found. If no matching privilege is found, Sentry returns `false` indicating "Access Denied" .

Managing Sentry for Impala through Cloudera Manager

To enable the Sentry service for Impala and Hive, set **Hive/Impala > Service-Wide > Sentry Service** parameter to the Sentry service. Then restart Impala and Hive. Simply adding Sentry service as a dependency and restarting enables Impala and Hive to use the Sentry service.

To set the server name to use when granting server level privileges, set the **Hive > Service-Wide > Advanced > Server Name for Sentry Authorization** parameter. When using Sentry with the Hive Metastore, you can specify the list of users that are allowed to bypass Sentry Authorization in Hive Metastore using **Hive > Service-Wide > Security > Bypass Sentry Authorization Users**. These are usually service users that already ensure all activity has been authorized.

> **Note:** The **Hive/Impala > Service-Wide > Policy File Based Sentry** tab contains parameters only relevant to configuring Sentry using policy files. In particular, make sure that **Enable Sentry Authorization using Policy Files** parameter is unchecked when using the Sentry service. Cloudera Manager throws a validation error if you attempt to configure the Sentry service and policy file at the same time.

The DEFAULT Database in a Secure Deployment

Because of the extra emphasis on granular access controls in a secure deployment, you should move any important or sensitive information out of the `DEFAULT` database into a named database whose privileges are specified in the policy file. Sometimes you might need to give privileges on the `DEFAULT` database for administrative reasons; for example, as a place you can reliably specify with a USE statement when preparing to drop a database.

Configuring Sentry Authorization for Cloudera Search

Sentry enables role-based, fine-grained authorization for Cloudera Search. Sentry can apply a range of restrictions to various actions, such as accessing data, managing configurations through config objects, or creating collections. Restrictions are consistently applied, regardless of how users attempt to complete actions. For example, restricting access to data in a collection restricts that access whether queries come from the command line, a browser, Hue, or through the admin console. For additional information on Sentry, see Authorization With Apache Sentry on page 346.

- This configuration process can be completed using either Cloudera Manager or the command-line instructions.
- This information applies specifically to CDH 5.13.x. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

Follow the instructions below to configure Sentry for Solr.

> **Note:** Sentry for Search depends on Kerberos authentication. For additional information on using Kerberos with Search, see Solr Authentication on page 151.

Setting Sentry Admins for Solr

If you are using the Sentry service (instead of a Sentry policy file), policies for Solr can be managed using the `solrctl sentry` command. To use this functionality, you must first designate a Sentry admin.

In Cloudera Manager:

1. Navigate to the Sentry service configuration page (**Sentry service** > **Configuration**).

2. In the **Admin Groups** field, add the name of a group to which you want to grant Sentry admin rights.
3. In the **Allowed Connecting Users** field, add the users to which you want to grant Sentry admin rights. These users must be members of at least one of the groups specified in the **Admin Groups** field.
4. Click **Save Changes**.
5. Restart the Sentry service (**Sentry service** > **Actions** > **Restart**).

If you are using the Sentry service without Cloudera Manager:

1. Edit `sentry-site.xml` file as follows:
    a. Add the Sentry admin group to the comma-separated list of groups in the `sentry.service.admin.group` property.
    b. Add the Sentry admin users to the comma-separated list of users in the `sentry.service.allow.connect` property.
2. Restart the Sentry service:

```
bin/sentry --command service --conffile /path/to/sentry-site.xml
```

Using Roles and Privileges with Sentry

Sentry uses a role-based privilege model. A role is assigned a set of rules for accessing a given Solr collection or Solr config. Access to each collection is controlled by three privileges: `Query`, `Update`, and `*`. The wildcard (`*`) privilege indicates all privileges.

The `admin` collection is a special collection used to represent administrative actions. A non-administrative request may only require privileges on the collection or config on which the request is being performed. An administrative request generally requires privileges on both the `admin` collection and the collection on which the action is being performed. For more information on the privilege model for Search, including a mapping of actions to privilege requirements, see Authorization Privilege Model for Solr.

In contrast, access to config objects is controlled by a single privilege, `*`, meaning all privileges.

You can also use the wildcard (`*`) to specify all config objects or collections when granting privileges. The following example syntax applies to both native Sentry privileges and file-based privileges, though native Sentry privileges are set by using `solrctl sentry` commands as shown in Using Solr with the Sentry Service on page 428, and file-based privileges are set in policy files as shown in Using Solr with a Policy File on page 429.

For example:

- A rule for the `Query` privilege on collection named `logs` is formulated as follows:

```
collection=logs->action=Query
```

- A rule for the `*` privilege, meaning all privileges, on the config named `myConfig` is formulated as follows:

```
config=myConfig->action=*
```

  No action implies `*`. Because `config` objects only support the `*` action, the following `config` privilege is invalid:

```
config=myConfig->action=Update
```

- A rule granting all collections the `Query` privilege is formulated as follows:

```
collection=*->action=Query
```

`config` objects cannot be combined with `collection` objects in a single privilege. For example, the following combinations are invalid:

```
config=myConfig->collection=myCollection->action=*
```

```
collection=myCollection->config=myConfig
```

You must specify these privileges separately. For example:

```
myRole = collection=myCollection->action=QUERY, config=myConfig->action=*
```

A role can contain multiple such rules, separated by commas. For example the `engineer_role` might contain the `Query` privilege for `hive_logs` and `hbase_logs` collections, and the `Update` privilege for the `current_bugs` collection. This example is formulated as follows:

```
engineer_role = collection=hive_logs->action=Query, collection=hbase_logs->action=Query,
 collection=current_bugs->action=Update
```

Using Users and Groups with Sentry

- A user is an entity that is permitted by the Kerberos authentication system to access the Search service.
- A group connects the authentication system with the authorization system. It is a set of one or more users who have been granted one or more authorization roles. Sentry allows a set of roles to be configured for a group.
- A configured group provider specifies how group membership is determined. Sentry supports HDFS-backed groups and locally configured groups. For example,

```
dev_ops = dev_role, ops_role
```

Here the group `dev_ops` is granted the roles `dev_role` and `ops_role`. The members of this group can perform the actions that are allowed by these roles.

User to Group Mapping

You can configure Sentry to use either Hadoop groups or groups defined in the policy file.

> **Important:** You can not use both Hadoop groups and local groups at the same time. Local groups are useful for proof-of-concept testing. For production environments, use Hadoop groups.

To configure Hadoop groups:

Set the `sentry.provider` property in `sentry-site.xml` to `org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider`.

By default, this uses local shell groups. See the Group Mapping section of the HDFS Permissions Guide for more information.

In this case, Sentry uses the Hadoop configuration described in Configuring LDAP Group Mappings on page 343. Cloudera Manager automatically uses this configuration. In a deployment not managed by Cloudera Manager, manually set these configuration parameters in the `hadoop-conf` file that is passed to Solr.

OR

To configure local groups:

1. Define local groups in a `[users]` section of the Sentry Policy file. For example:

```
[users]
user1 = group1, group2, group3
user2 = group2, group3
```

2. In `sentry-site.xml`, set `search.sentry.provider` as follows:

```
<property>
    <name>sentry.provider</name>
   <value>org.apache.sentry.provider.file.LocalGroupResourceAuthorizationProvider</value>

  </property>
```

Enabling Caching for the Sentry Service

Using the Sentry Service with Cloudera Search can introduce latency because authorization requests must be sent to the Sentry Service. To alleviate this latency, enable caching by adding the following property to `sentry-site.xml` on each Solr Server:

```
<property>
  <name>sentry.provider.backend.generic.cache.enabled</name>
  <value>true</value>
</property>
```

By default, this caches Sentry responses for 30 seconds. To modify the cache duration, add the following property to `sentry-site.xml` on each Solr Server:

```
<property>
  <name>sentry.provider.backend.generic.cache.ttl.ms</name>
  <value>30000</value>
</property>
```

The value is set in milliseconds.

For Cloudera Manager environments, add these properties to the **Advanced Configuration Snippet** for `sentry-site.xml`:

1. Go to **Solr service** > **Configuration** > **Advanced** > **Solr Service Advanced Configuration Snippet (Safety Valve) for sentry-site.xml**.
2. Click the **Add** button.
3. Enter the following values:
   - **Name:** `sentry.provider.backend.generic.cache.enabled`
   - **Value:** `true`
4. Click the **Add** button.
5. Enter the following values:
   - **Name:** `sentry.provider.backend.generic.cache.ttl.ms`
   - **Value:** `30000`
6. Click **Save Changes**.
7. Restart the Solr service (**Solr service** > **Actions** > **Restart**).

Sample Sentry Configuration

This section provides sample configurations.

Using Solr with the Sentry Service

In CDH 5.8, Cloudera Search adds support for storing permissions in the Sentry service. You can enable storing permissions in the Sentry service by Enabling the Sentry Service for Solr on page 366. If you have already configured Sentry's policy file-based approach, you can migrate existing authorization settings as described in Migrating from Sentry Policy Files to the Sentry Service on page 363. `solrctl` has been extended to support:

- Migrating existing policy files to the Sentry service
- Managing managing permissions in the Sentry service

The following is an example of the commands used to configure Sentry for Solr using `solrctl sentry` command. These commands should be run on a host with a Solr **Gateway** role.

These sample commands that follow illustrate establishing two different roles, each of which have different access requirements. The process of creating roles, adding roles to groups, and granting privileges to roles is a typical workflow used to provide different groups varied degrees of access to resources. For reference information, see solrctl Reference.

Begin by creating roles. The following command creates `ops_role` and `dev_ops_role`:

```
solrctl sentry --create-role ops_role
solrctl sentry --create-role dev_ops_role
```

Next, add existing Hadoop groups to the roles you created. The following command adds `ops_role` to the existing `ops_group` Hadoop group and adds `dev_ops_role` to the existing `dev_ops_group` Hadoop group:

```
solrctl sentry --add-role-group ops_role ops_group
solrctl sentry --add-role-group dev_ops_role dev_ops_group
```

Finally, add privileges to collections and configs to roles. The following command adds the QUERY privilege to `ops_role` for the `logs` collection and all privileges (meaning QUERY and UPDATE) to the `dev_ops_role` for all (`*`) collections:

```
solrctl sentry --grant-privilege ops_role 'collection=logs->action=Query'
solrctl sentry --grant-privilege dev_ops_role 'collection=*->action=*'
```

Using Solr with a Policy File

Use separate policy files for each Sentry-enabled service. Using one file for multiple services results in each service failing on the other services' entries. For example, with a combined Hive and Search file, Search would fail on Hive entries and Hive would fail on Search entries.

Sentry with Search does not support multiple policy files. Other implementations of Sentry such as Sentry for Hive do support different policy files for different databases, but Sentry for Search has no such support for multiple policies.

The following is an example of a Search policy file. The This location must be readable by Solr.

sentry-provider.ini

```
[groups]
# Assigns each Hadoop group to its set of roles
engineer = engineer_role
ops = ops_role
dev_ops = engineer_role, ops_role
hbase_admin = hbase_admin_role

[roles]
# The following grants all access to source_code.
# "collection = source_code" can also be used as syntactic
# sugar for "collection = source_code->action=*"
engineer_role = collection = source_code->action=*

# The following imply more restricted access.
ops_role = collection = hive_logs->action=Query
dev_ops_role = collection = hbase_logs->action=Query

#give hbase_admin_role the ability to create/delete/modify the hbase_logs collection
#as well as to update the config for the hbase_logs collection, called hbase_logs_config.
hbase_admin_role = collection=admin->action=*, collection=hbase_logs->action=*,
config=hbase_logs_config->action=*
```

Sentry Configuration File

Sentry can store configuration as well as privilege policies in files. The `sentry-site.xml` file contains configuration options such as privilege policy file location. The policy files contains the privileges and groups. It has a `.ini` file format and should be stored on HDFS.

The following is an example of a `sentry-site.xml` file.

sentry-site.xml

```
<configuration>
  <property>
    <name>hive.sentry.provider</name>

<value>org.apache.sentry.provider.file.HadoopGroupResourceAuthorizationProvider</value>

  </property>

  <property>
    <name>sentry.solr.provider.resource</name>
    <value>/path/to/authz-provider.ini</value>
    <!--
        If the HDFS configuration files (core-site.xml, hdfs-site.xml)
        pointed to by SOLR_HDFS_CONFIG in /etc/default/solr
        point to HDFS, the path will be in HDFS;
        alternatively you could specify a full path,
        e.g.:hdfs://namenode:port/path/to/authz-provider.ini
    -->
  </property>
```

Using Policy Files with Sentry

This section contains notes on creating and maintaining the policy file.

Storing the Policy File

Considerations for storing the policy file(s) include:

1. Replication count - Because Sentry reads the file for each query, you should increase this. 10 is a reasonable value.
2. Updating the file - Updates to the file are only reflected when the Solr process is restarted.

Defining Roles

Keep in mind that role definitions are not cumulative. The newer definition replaces the older one. For example, consider the following definition:

```
role1 = privilege1
role1 = privilege2
```

This definition results in `role1` having `privilege2`, not `privilege1` and `privilege2`.

Providing Document-Level Security Using Sentry

For role-based access control of a collection, an administrator modifies a Sentry role so it has query, update, or administrative access.

Collection-level authorization is useful when the access control requirements for the documents in the collection are the same, but users may want to restrict access to a subset of documents in a collection. This finer-grained restriction can be achieved by defining separate collections for each subset, but this is difficult to manage, requires duplicate documents for each collection, and requires that these documents be kept synchronized.

Document-level access control solves this issue by associating authorization tokens with each document in the collection. This enables granting Sentry roles access to sets of documents in a collection.

Document-Level Security Model

Document-level security depends on a chain of relationships between users, groups, roles, and documents.

- Users are assigned to groups.
- Groups are assigned to roles.
- Roles are stored as "authorization tokens" in a specified field in the documents.

Document-level security supports restricting which documents can be viewed by which users. Access is provided by adding roles as "authorization tokens" to a specified document field. Conversely, access is implicitly denied by omitting roles from the specified field. In other words, in a document-level security enabled environment, a user might submit a query that matches a document; if the user is not part of a group that has a role has been granted access to the document, the result is not returned.

For example, Alice might belong to the administrators group. The administrators group may belong to the doc-mgmt role. A document could be ingested and the doc-mgmt role could be added at ingest time. In such a case, if Alice submitted a query that matched the document, Search would return the document, since Alice is then allowed to see any document with the "doc-mgmt" authorization token.

Similarly, Bob might belong to the guests group. The guests group may belong to the public-browser role. If Bob tried the same query as Alice, but the document did not have the public-browser role, Search would not return the result because Bob does not belong to a group that is associated with a role that has access.

Note that collection-level authorization rules still apply, if enabled. Even if Alice is able to view a document given document-level authorization rules, if she is not allowed to query the collection, the query will fail.

Roles are typically added to documents when those documents are ingested, either using the standard Solr APIs or, if using morphlines, the `setValues` morphline command.

Enabling Document-Level Security

Cloudera Search supports document-level security in Search for CDH 5.1 and higher. Document-level security requires collection-level security. Configuring collection-level security is described earlier in this topic.

Document-level security is disabled by default, so the first step in using document-level security is to enable the feature by modifying the `solrconfig.xml.secure` file. Remember to replace the `solrconfig.xml` with this file, as described in Enabling Solr as a Client for the Sentry Service Using the Command Line on page 372.

To enable document-level security, change `solrconfig.xml.secure`. The default file contents are as follows:

```
<searchComponent name="queryDocAuthorization">
    <!-- Set to true to enabled document-level authorization -->

    <bool name="enabled">false</bool>


    <!-- Field where the auth tokens are stored in the document -->
    <str name="sentryAuthField">sentry_auth</str>


    <!-- Auth token defined to allow any role to access the document.
         Uncomment to enable. -->

    <!--<str name="allRolesToken">*</str>-->

</searchComponent>
```

- The `enabled` Boolean determines whether document-level authorization is enabled. To enable document level security, change this setting to `true`.
- The `sentryAuthField` string specifies the name of the field that is used for storing authorization information. You can use the default setting of `sentry_auth` or you can specify some other string to be used for assigning values during ingest.

> **Note:** This field must exist as an explicit or dynamic field in the schema for the collection you are creating with document-level security. `sentry_auth` exists in the default `schema.xml`, which is automatically generated and can be found in the same directory as `solrconfig.xml`.
>
> for the collection you are creating with document-level security. Schema.xml is in the generated configuration in the same directory as the solrconfig.xml

- The `allRolesToken` string represents a special token defined to allow any role access to the document. By default, this feature is disabled. To enable this feature, uncomment the specification and specify the token. This token should be different from the name of any sentry role to avoid collision. By default it is "*". This feature is useful when first configuring document level security or it can be useful in granting all roles access to a document when the set of roles may change. See Best Practices on page 432 for additional information.

Best Practices

Using `allRolesToken`

You may want to grant every user that belongs to a role access to certain documents. One way to accomplish this is to specify all known roles in the document, but this requires updating or re-indexing the document if you add a new role. Alternatively, an `allUser` role, specified in the Sentry `.ini` file, could contain all valid groups, but this role would need to be updated every time a new group was added to the system. Instead, specifying `allRolesToken` allows any user that belongs to a valid role to access the document. This access requires no updating as the system evolves.

In addition, `allRolesToken` may be useful for transitioning a deployment to use document-level security. Instead of having to define all the roles upfront, all the documents can be specified with `allRolesToken` and later modified as the roles are defined.

Consequences of Document-Level Authorization Only Affecting Queries

Document-level security does not prevent users from modifying documents or performing other update operations on the collection. Update operations are only governed by collection-level authorization.

Document-level security can be used to prevent documents being returned in query results. If users are not granted access to a document, those documents are not returned even if that user submits a query that matches those documents. This does not have affect attempted updates.

Consequently, it is possible for a user to not have access to a set of documents based on document-level security, but to still be able to modify the documents using their collection-level authorization update rights. This means that a user can delete all documents in the collection. Similarly, a user might modify all documents, adding their authorization token to each one. After such a modification, the user could access any document using querying. Therefore, if you are restricting access using document-level security, consider granting collection-level update rights only to those users you trust and assume they will be able to access every document in the collection.

Limitations on Query Size

By default queries support up to 1024 Boolean clauses. As a result, queries containing more that 1024 clauses may cause errors. Because authorization information is added by Sentry as part of a query, using document-level security can increase the number of clauses. In the case where users belong to many roles, even simple queries can become quite large. If a query is too large, an error of the following form occurs:

```
org.apache.lucene.search.BooleanQuery$TooManyClauses: maxClauseCount is set to 1024
```

To change the supported number of clauses, edit the `maxBooleanClauses` setting in `solrconfig.xml`. For example, to allow 2048 clauses, you would edit the setting so it appears as follows:

```
<maxBooleanClauses>2048</maxBooleanClauses>
```

For `maxBooleanClauses` to be applied as expected, make any change to this value to all collections and then restart the service. You must make this change to all collections because this option modifies a global Lucene property, affecting all Solr cores. If different `solrconfig.xml` files have different values for this property, the effective value is determined per host, based on the first Solr core to be initialized.

Enabling Secure Impersonation

Secure impersonation allows a user to make requests as another user in a secure way. The user who has been granted impersonation rights receives the same access as the user being impersonated.

Configure custom security impersonation settings using the **Solr Service Environment Advanced Configuration Snippet (Safety Valve)**. For example, to allow the following impersonations:

- User hue can make requests as any user from any host.
- User `foo` can make requests as any member of group `bar`, from `host1` or `host2`.

Enter the following values into the **Solr Service Environment Advanced Configuration Snippet (Safety Valve)**:

```
SOLR_SECURITY_ALLOWED_PROXYUSERS=hue,foo
SOLR_SECURITY_PROXYUSER_hue_HOSTS=*
SOLR_SECURITY_PROXYUSER_hue_GROUPS=*
SOLR_SECURITY_PROXYUSER_foo_HOSTS=host1,host2
SOLR_SECURITY_PROXYUSER_foo_GROUPS=bar
```

`SOLR_SECURITY_ALLOWED_PROXYUSERS` lists all of the users allowed to impersonate. For a user x in `SOLR_SECURITY_ALLOWED_PROXYUSERS`, `SOLR_SECURITY_PROXYUSER_x_HOSTS` list the hosts x is allowed to connect from to impersonate, and `SOLR_SECURITY_PROXYUSERS_x_GROUPS` lists the groups that the users is allowed to impersonate members of. Both `GROUPS` and `HOSTS` support the wildcard * and both `GROUPS` and `HOSTS` must be defined for a specific user.

> **Note:** Cloudera Manager has its own management of secure impersonation for Hue. To add additional users for Secure Impersonation, use the environment variable safety value for Solr to set the environment variables as above. Be sure to include hue in `SOLR_SECURITY_ALLOWED_PROXYUSERS` if you want to use secure impersonation for hue.

Configuring HBase Authorization

> **Warning:** Disabling security on a production HBase system is difficult and could cause data loss. Contact Cloudera Support if you need to disable security in HBase.

After configuring HBase authentication (as detailed in HBase Configuration), you must define rules on resources that is allowed to access. HBase rules can be defined individual tables, columns, and cells within a table. Cell-level authorization was added as an experimental feature in CDH 5.2 and is still considered experimental.

Understanding HBase Access Levels

HBase access levels are granted independently of each other and allow for different types of operations at a given scope.

- **Read (R)** - can read data at the given scope
- **Write (W)** - can write data at the given scope
- **Execute (X)** - can execute coprocessor endpoints at the given scope
- **Create (C)** - can create tables or drop tables (even those they did not create) at the given scope
- **Admin (A)** - can perform cluster operations such as balancing the cluster or assigning regions at the given scope

The possible scopes are:

- **Superuser** - superusers can perform any operation available in HBase, to any resource. The user who runs HBase on your cluster is a superuser, as are any principals assigned to the configuration property `hbase.superuser` in `hbase-site.xml` on the HMaster.
- **Global** - permissions granted at `global` scope allow the admin to operate on all tables of the cluster.
- **Namespace** - permissions granted at `namespace` scope apply to all tables within a given namespace.
- **Table** - permissions granted at `table` scope apply to data or metadata within a given table.
- **ColumnFamily** - permissions granted at `ColumnFamily` scope apply to cells within that ColumnFamily.
- **Cell** - permissions granted at Cell scope apply to that exact cell coordinate. This allows for policy evolution along with data. To change an ACL on a specific cell, write an updated cell with new ACL to the precise coordinates of the original. If you have a multi-versioned schema and want to update ACLs on all visible versions, you'll need to write new cells for all visible versions. The application has complete control over policy evolution. The exception is append and `increment` processing. Appends and `increments` can carry an ACL in the operation. If one is included in the operation, then it will be applied to the result of the append or `increment`. Otherwise, the ACL of the existing cell being appended to or incremented is preserved.

The combination of access levels and scopes creates a matrix of possible access levels that can be granted to a user. In a production environment, it is useful to think of access levels in terms of what is needed to do a specific job. The following list describes appropriate access levels for some common types of HBase users. It is important not to grant more access than is required for a given user to perform their required tasks.

- **Superusers** - In a production system, only the HBase user should have superuser access. In a development environment, an administrator might need superuser access to quickly control and manage the cluster. However, this type of administrator should usually be a `Global Admin` rather than a superuser.
- **Global Admins** - A `global admin` can perform tasks and access every table in HBase. In a typical production environment, an admin should not have `Read` or `Write` permissions to data within tables.
  - A global admin with `Admin` permissions can perform cluster-wide operations on the cluster, such as balancing, assigning or unassigning regions, or calling an explicit major compaction. This is an operations role.
  - A global admin with `Create` permissions can create or drop any table within HBase. This is more of a DBA-type role.

  In a production environment, it is likely that different users will have only one of `Admin` and `Create` permissions.

> **Warning:**
>
> In the current implementation, a `Global Admin` with `Admin` permission can grant himself `Read` and `Write` permissions on a table and gain access to that table's data. For this reason, only grant `Global Admin` permissions to trusted user who actually need them.
>
> Also be aware that a `Global Admin` with `Create` permission can perform a `Put` operation on the ACL table, simulating a `grant` or `revoke` and circumventing the authorization check for `Global Admin` permissions. This issue (but not the first one) is fixed in CDH 5.3 and higher, as well as CDH 5.2.1.
>
> Due to these issues, be cautious with granting Global Admin privileges.

- **Namespace Admin** - a namespace admin with Create permissions can create or drop tables within that namespace, and take and restore snapshots. A namespace admin with Admin permissions can perform operations such as splits or major compactions on tables within that namespace. Prior to CDH 5.4, only global admins could create namespaces. In CDH 5.4, any user with Namespace Create privileges can create namespaces.
- **Table Admins** - A table admin can perform administrative operations only on that table. A table admin with `Create` permissions can create snapshots from that table or restore that table from a snapshot. A table admin with `Admin` permissions can perform operations such as splits or major compactions on that table.
- **Users** - Users can read or write data, or both. Users can also execute coprocessor endpoints, if given `Executable` permissions.

> **Important:**
>
> If you are using Kerberos principal names when setting ACLs for users, Hadoop uses only the first part (short) of the Kerberos principal when converting it to the username. Hence, for the principal `ann/fully.qualified.domain.name@YOUR-REALM.COM`, HBase ACLs should only be set for user `ann`.

The following table shows some typical job descriptions at a hypothetical company and the permissions they might require to get their jobs done using HBase.

Table 49: Real-World Example of Access Levels

Job Title	Scope	Permissions	Description
Senior Administrator	Global	Admin, Create	Manages the cluster and gives access to Junior Administrators.
Junior Administrator	Global	Create	Creates tables and gives access to Table Administrators.
Table Administrator	Table	Admin	Maintains a table from an operations point of view.
Data Analyst	Table	Read	Creates reports from HBase data.
Web Application	Table	Read, Write	Puts data into HBase and uses HBase data to perform operations.

Further Reading

- Access Control Matrix
- Security - Apache HBase Reference Guide

Enable HBase Authorization

HBase authorization is built on top of the Coprocessors framework, specifically `AccessController Coprocessor`.

> **Note:** Once the Access Controller coprocessor is enabled, any user who uses the HBase shell will be subject to access control. Access control will also be in effect for native (Java API) client access to HBase.

Enable HBase Authorization Using Cloudera Manager

1. Go to **Clusters** and select the HBase cluster.
2. Select **Configuration**.
3. Search for **HBase Secure Authorization** and select it.
4. Search for **HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml** and enter the following into it to enable `hbase.security.exec.permission.checks`. Without this option, all users will continue to have access to execute endpoint coprocessors. This option is not enabled when you enable HBase Secure Authorization for backward compatibility.

```
<property>
  <name>hbase.security.exec.permission.checks</name>
  <value>true</value>
</property>
```

5. Optionally, search for and configure **HBase Coprocessor Master Classes** and **HBase Coprocessor Region Classes**.

Enable HBase Authorization Using the Command Line

> **Important:**
>
> - Follow these command-line instructions on systems that do not use Cloudera Manager.
> - This information applies specifically to CDH 5.13.x. See Cloudera Documentation for information specific to other releases.

To enable HBase authorization, add the following properties to the `hbase-site.xml` file *on every HBase server host (Master or RegionServer)*:

```
<property>
     <name>hbase.security.authorization</name>
     <value>true</value>
</property>
<property>
  <name>hbase.security.exec.permission.checks</name>
  <value>true</value>
</property>
<property>
     <name>hbase.coprocessor.master.classes</name>
     <value>org.apache.hadoop.hbase.security.access.AccessController</value>
</property>
<property>
     <name>hbase.coprocessor.region.classes</name>

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

Configure Access Control Lists for Authorization

Now that HBase has the security coprocessor enabled, you can set ACLs using the HBase shell. Start the HBase shell as usual.

> **Important:**
>
> The host running the shell must be configured with a keytab file as described in Configuring Kerberos Authentication for HBase.

The commands that control ACLs take the following form. Group names are prefixed with the @ symbol.

```
hbase> grant <user> <permissions> [ @<namespace> [ <table>[ <column family>[ <column
qualifier> ] ] ] ]   # grants permissions

hbase> revoke <user> <permissions> [ @<namespace> [ <table> [ <column family> [ <column
 qualifier> ] ] ]  # revokes permissions

hbase> user_permission <table>
                    # displays existing permissions
```

In the above commands, fields encased in <> are variables, and fields in [] are optional. The `permissions` variable must consist of zero or more character from the set "RWCA".

- R denotes read permissions, which is required to perform `Get`, `Scan`, or `Exists` calls in a given scope.
- `W` denotes write permissions, which is required to perform `Put`, `Delete`, `LockRow`, `UnlockRow`, `IncrementColumnValue`, `CheckAndDelete`, `CheckAndPut`, `Flush`, or `Compact` in a given scope.
- X denotes execute permissions, which is required to execute coprocessor endpoints.
- C denotes create permissions, which is required to perform `Create`, `Alter`, or `Drop` in a given scope.
- A denotes admin permissions, which is required to perform `Enable`, `Disable`, `Snapshot`, `Restore`, `Clone`, `Split`, `MajorCompact`, `Grant`, `Revoke`, and `Shutdown` in a given scope.

Access Control List Example Commands

```
grant 'user1', 'RWC'
grant 'user2', 'RW', 'tableA'
grant 'user3', 'C', '@my_namespace'
```

Be sure to review the information in Understanding HBase Access Levels on page 433 to understand the implications of the different access levels.

Impala Security Overview

Impala includes a fine-grained authorization framework for Hadoop, based on the Sentry open source project. Sentry authorization was added in Impala 1.1.0. Together with the Kerberos authentication framework, Sentry takes Hadoop security to a new level needed for the requirements of highly regulated industries such as healthcare, financial services, and government. Impala also includes an auditing capability; Impala generates the audit data, the Cloudera Navigator product consolidates the audit data from all nodes in the cluster, and Cloudera Manager lets you filter, visualize, and produce reports. The auditing feature was added in Impala 1.1.1.

The Impala security features have several objectives. At the most basic level, security prevents accidents or mistakes that could disrupt application processing, delete or corrupt data, or reveal data to unauthorized users. More advanced security features and practices can harden the system against malicious users trying to gain unauthorized access or perform other disallowed operations. The auditing feature provides a way to confirm that no unauthorized access occurred, and detect whether any such attempts were made. This is a critical set of features for production deployments in large organizations that handle important or sensitive data. It sets the stage for multi-tenancy, where multiple applications run concurrently and are prevented from interfering with each other.

The material in this section presumes that you are already familiar with administering secure Linux systems. That is, you should know the general security practices for Linux and Hadoop, and their associated commands and configuration files. For example, you should know how to create Linux users and groups, manage Linux group membership, set Linux and HDFS file permissions and ownership, and designate the default permissions and ownership for new files. You should be familiar with the configuration of the nodes in your Hadoop cluster, and know how to apply configuration changes or run a set of commands across all the nodes.

The security features are divided into these broad categories:

authorization

Which users are allowed to access which resources, and what operations are they allowed to perform? Impala relies on the open source Sentry project for authorization. By default (when authorization is not enabled), Impala does all read and write operations with the privileges of the `impala` user, which is suitable for a development/test environment but not for a secure production environment. When authorization is enabled, Impala uses the OS user ID of the user who runs `impala-shell` or other client program, and associates various privileges with each user. See Enabling Sentry Authorization for Impala on page 415 for details about setting up and managing authorization.

authentication

How does Impala verify the identity of the user to confirm that they really are allowed to exercise the privileges assigned to that user? Impala relies on the Kerberos subsystem for authentication. See Enabling Kerberos Authentication for Impala on page 141 for details about setting up and managing authentication.

auditing

What operations were attempted, and did they succeed or not? This feature provides a way to look back and diagnose whether attempts were made to perform unauthorized operations. You use this information to track down suspicious activity, and to see where changes are needed in authorization policies. The audit data produced by this feature is collected by the Cloudera Manager product and then presented in a user-friendly form by the Cloudera Manager product. See Auditing Impala Operations for details about setting up and managing auditing.

These other topics in the *Security Guide* cover how Impala integrates with security frameworks such as Kerberos, LDAP, and Sentry:

- Impala Authentication on page 141
- Enabling Sentry Authorization for Impala on page 415

Security Guidelines for Impala

The following are the major steps to harden a cluster running Impala against accidents and mistakes, or malicious attackers trying to access sensitive data:

- Secure the `root` account. The `root` user can tamper with the `impalad` daemon, read and write the data files in HDFS, log into other user accounts, and access other system services that are beyond the control of Impala.
- Restrict membership in the `sudoers` list (in the `/etc/sudoers` file). The users who can run the `sudo` command can do many of the same things as the `root` user.
- Ensure the Hadoop ownership and permissions for Impala data files are restricted.
- Ensure the Hadoop ownership and permissions for Impala log files are restricted.
- Ensure that the Impala web UI (available by default on port 25000 on each Impala node) is password-protected. See [Impala Web User Interface for Debugging]() for details.
- Create a policy file that specifies which Impala privileges are available to users in particular Hadoop groups (which by default map to Linux OS groups). Create the associated Linux groups using the `groupadd` command if necessary.
- The Impala authorization feature makes use of the HDFS file ownership and permissions mechanism; for background information, see the [HDFS Permissions Guide](). Set up users and assign them to groups at the OS level, corresponding to the different categories of users with different access levels for various databases, tables, and HDFS locations (URIs). Create the associated Linux users using the `useradd` command if necessary, and add them to the appropriate groups with the `usermod` command.
- Design your databases, tables, and views with database and table structure to allow policy rules to specify simple, consistent rules. For example, if all tables related to an application are inside a single database, you can assign privileges for that database and use the `*` wildcard for the table name. If you are creating views with different privileges than the underlying base tables, you might put the views in a separate database so that you can use the `*` wildcard for the database containing the base tables, while specifying the precise names of the individual views. (For specifying table or database names, you either specify the exact name or `*` to mean all the databases on a server, or all the tables and views in a database.)
- Enable authorization by running the `impalad` daemons with the `-server_name` and `-authorization_policy_file` options on all nodes. (The authorization feature does not apply to the `statestored` daemon, which has no access to schema objects or data files.)
- Set up authentication using Kerberos, to make sure users really are who they say they are.

Securing Impala Data and Log Files

One aspect of security is to protect files from unauthorized access at the filesystem level. For example, if you store sensitive data in HDFS, you specify permissions on the associated files and directories in HDFS to restrict read and write permissions to the appropriate users and groups.

If you issue queries containing sensitive values in the `WHERE` clause, such as financial account numbers, those values are stored in Impala log files in the Linux filesystem and you must secure those files also. For the locations of Impala log files, see [Using Impala Logging]().

All Impala read and write operations are performed under the filesystem privileges of the `impala` user. The `impala` user must be able to read all directories and data files that you query, and write into all the directories and data files for `INSERT` and `LOAD DATA` statements. At a minimum, make sure the `impala` user is in the `hive` group so that it can access files and directories shared between Impala and Hive. See [User Account Requirements]() for more details.

Setting file permissions is necessary for Impala to function correctly, but is not an effective security practice by itself:

- The way to ensure that only authorized users can submit requests for databases and tables they are allowed to access is to set up Sentry authorization, as explained in [Enabling Sentry Authorization for Impala]() on page 415. With authorization enabled, the checking of the user ID and group is done by Impala, and unauthorized access is blocked by Impala itself. The actual low-level read and write requests are still done by the `impala` user, so you must have appropriate file and directory permissions for that user ID.

- You must also set up Kerberos authentication, as described in Enabling Kerberos Authentication for Impala on page 141, so that users can only connect from trusted hosts. With Kerberos enabled, if someone connects a new host to the network and creates user IDs that match your privileged IDs, they will be blocked from connecting to Impala at all from that host.

Installation Considerations for Impala Security

Impala 1.1 comes set up with all the software and settings needed to enable security when you run the `impalad` daemon with the new security-related options (`-server_name` and `-authorization_policy_file`). You do not need to change any environment variables or install any additional JAR files. In a cluster managed by Cloudera Manager, you do not need to change any settings in Cloudera Manager.

Securing the Hive Metastore Database

It is important to secure the Hive metastore, so that users cannot access the names or other information about databases and tables the through the Hive client or by querying the metastore database. Do this by turning on Hive metastore security, using the instructions in the CDH 5 Security Guide for securing different Hive components:

- Secure the Hive Metastore.
- In addition, allow access to the metastore only from the HiveServer2 server, and then disable local access to the HiveServer2 server.

Securing the Impala Web User Interface

The instructions in this section presume you are familiar with the `.htpasswd` mechanism commonly used to password-protect pages on web servers.

Password-protect the Impala web UI that listens on port 25000 by default. Set up a `.htpasswd` file in the `$IMPALA_HOME` directory, or start both the `impalad` and `statestored` daemons with the `--webserver_password_file` option to specify a different location (including the filename).

This file should only be readable by the Impala process and machine administrators, because it contains (hashed) versions of passwords. The username / password pairs are not derived from Unix usernames, Kerberos users, or any other system. The `domain` field in the password file must match the domain supplied to Impala by the new command-line option `--webserver_authentication_domain`. The default is `mydomain.com`.

Impala also supports using HTTPS for secure web traffic. To do so, set `--webserver_certificate_file` to refer to a valid `.pem` TLS/SSL certificate file. Impala will automatically start using HTTPS once the TLS/SSL certificate has been read and validated. A `.pem` file is basically a private key, followed by a signed TLS/SSL certificate; make sure to concatenate both parts when constructing the `.pem` file.

If Impala cannot find or parse the `.pem` file, it prints an error message and quits.

> **Note:**
>
> If the private key is encrypted using a passphrase, Impala will ask for that passphrase on startup, which is not useful for a large cluster. In that case, remove the passphrase and make the `.pem` file readable only by Impala and administrators.
>
> When you turn on TLS/SSL for the Impala web UI, the associated URLs change from `http://` prefixes to `https://`. Adjust any bookmarks or application code that refers to those URLs.

Kudu Security Overview

Kudu includes security features that allow Kudu clusters to be hardened against access from unauthorized users. Kudu uses strong authentication with Kerberos, while communication between Kudu clients and servers can now be encrypted with TLS. Kudu also allows you to use HTTPS encryption to connect to the web UI.

The rest of this topic describes the security capabilities of Apache Kudu and how to configure a secure Kudu cluster. Currently, there are a few known limitations in Kudu security that might impact your cluster. For the list, see Security Limitations.

Kudu Authentication with Kerberos

Kudu can be configured to enforce secure authentication among servers, and between clients and servers. Authentication prevents untrusted actors from gaining access to Kudu, and securely identifies connecting users or services for authorization checks. Authentication in Kudu is designed to interoperate with other secure Hadoop components by utilizing Kerberos.

Configure authentication on Kudu servers using the `--rpc-authentication` flag, which can be set to one of the following options:

- `required` - Kudu will reject connections from clients and servers who lack authentication credentials.
- `optional` - Kudu will attempt to use strong authentication, but will allow unauthenticated connections.
- `disabled` - Kudu will only allow unauthenticated connections.

By default, the flag is set to `optional`. To secure your cluster, set `--rpc-authentication` to `required`.

Internal Private Key Infrastructure (PKI)

Kudu uses an internal PKI to issue X.509 certificates to servers in the cluster. Connections between peers who have both obtained certificates will use TLS for authentication. In such cases, neither peer needs to contact the Kerberos KDC.

X.509 certificates are only used for internal communication among Kudu servers, and between Kudu clients and servers. These certificates are never presented in a public facing protocol. By using internally-issued certificates, Kudu offers strong authentication which scales to huge clusters, and allows TLS encryption to be used without requiring you to manually deploy certificates on every node.

Authentication Tokens

After authenticating to a secure cluster, the Kudu client will automatically request an authentication token from the Kudu master. An authentication token encapsulates the identity of the authenticated user and carries the Kudu master's RSA signature so that its authenticity can be verified. This token will be used to authenticate subsequent connections. By default, authentication tokens are only valid for seven days, so that even if a token were compromised, it cannot be used indefinitely. For the most part, authentication tokens should be completely transparent to users. By using authentication tokens, Kudu is able to take advantage of strong authentication, without paying the scalability cost of communicating with a central authority for every connection.

When used with distributed compute frameworks such as Apache Spark, authentication tokens can simplify configuration and improve security. For example, the Kudu Spark connector will automatically retrieve an authentication token during the planning stage, and distribute the token to tasks. This allows Spark to work against a secure Kudu cluster where only the planner node has Kerberos credentials.

Scalability

Kudu authentication is designed to scale to thousands of nodes, which means it must avoid unnecessary coordination with a central authentication authority (such as the Kerberos KDC) for each connection. Instead, Kudu servers and clients use Kerberos to establish initial trust with the Kudu master, and then use alternate credentials for subsequent connections. As described previously, the Kudu master issues internal X.509 certificates to tablet servers on startup, and temporary authentication tokens to clients on first contact.

Encryption

Kudu allows you to use TLS to encrypt all communications among servers, and between clients and servers. Configure TLS encryption on Kudu servers using the `--rpc-encryption` flag, which can be set to one of the following options:

- `required` - Kudu will reject unencrypted connections.
- `optional` - Kudu will attempt to use encryption, but will allow unencrypted connections.
- `disabled` - Kudu will not use encryption.

By default, the flag is set to `optional`. To secure your cluster, set `--rpc-encryption` to `required`.

> **Note:** Kudu will automatically turn off encryption on local loopback connections, since traffic from these connections is never exposed externally. This allows locality-aware compute frameworks, such as Spark and Impala, to avoid encryption overhead, while still ensuring data confidentiality.

Coarse-grained Authorization

Kudu supports coarse-grained authorization checks for client requests based on the client's authenticated Kerberos principal (user or service). Access levels are granted based on whitelist-style Access Control Lists (ACLs), one for each level. Each ACL specifies a comma-separated list of users, or may be set to `'*'` to indicate that all authenticated users have access rights at the specified level.

The two levels of access which can be configured are:

- **Superuser** - Principals authorized as a superuser can perform certain administrative functions such as using the `kudu` command line tool to diagnose and repair cluster issues.
- **User** - Principals authorized as a user are able to access and modify all data in the Kudu cluster. This includes the ability to create, drop, and alter tables, as well as read, insert, update, and delete data. The default value for the User ACL is `'*'`, which allows all users access to the cluster. However, if authentication is enabled, this will restrict access to only those users who are able to successfully authenticate using Kerberos. Unauthenticated users on the same network as the Kudu servers will be unable to access the cluster.

> **Note:** Internally, Kudu has a third access level for the daemons themselves called **Service**. This is used to ensure that users cannot connect to the cluster and pose as tablet servers.

Web UI Encryption

The Kudu web UI can be configured to use secure HTTPS encryption by providing each server with TLS certificates. Use the `--webserver-certificate-file` and `--webserver-private-key-file` properties to specify the certificate and private key to be used for communication.

Alternatively, you can choose to completely disable the web UI by setting `--webserver-enabled` flag to `false` on the Kudu servers.

Web UI Redaction

To prevent sensitive data from being included in the web UI, all row data is redacted. Table metadata, such as table names, column names, and partitioning information is not redacted. Alternatively, you can choose to completely disable the web UI by setting the `--webserver-enabled` flag to `false` on the Kudu servers.

> **Note:** Disabling the web UI will also disable REST endpoints such as `/metrics`. Monitoring systems rely on these endpoints to gather metrics data.

Log Redaction

To prevent sensitive data from being included in Kudu server logs, all row data will be redacted. You can turn off log redaction using the `--redact` flag.

Configuring a Secure Kudu Cluster using Cloudera Manager

> **Warning:** If you are upgrading from Kudu 1.2.0 / CDH 5.10.x, you must upgrade both Kudu and CDH parcels (or packages) at the same time. If you upgrade Kudu but do not upgrade CDH, new Kudu features such as Security will not be available. Note that even though you might be able to see the updated configuration options for Kudu security in Cloudera Manager, configuring them will have no effect.

Use the following set of instructions to secure a Kudu cluster using Cloudera Manager:

Enabling Kerberos Authentication and RPC Encryption

> **Important:** The following instructions assume you already have a secure Cloudera Manager cluster with Kerberos authentication enabled. If this is not the case, first secure your cluster using the steps described at Enabling Kerberos Authentication Using the Cloudera Manager Wizard.

To enable Kerberos authentication for Kudu:

1. Go to the **Kudu** service.
2. Click the **Configuration** tab.
3. Select **Category** > **Main**.
4. In the Search field, type **Kerberos** to show the relevant properties.
5. Edit the following properties according to your cluster configuration:

Field	Usage Notes
Kerberos Principal	Set to the default principal, kudu. Currently, Kudu does not support configuring a custom service principal for Kudu processes.
Enable Secure Authentication And Encryption	Select this checkbox to enable authentication and RPC encryption between all Kudu clients and servers, as well as between individual servers. Only enable this property after you have configured Kerberos.

6. Click **Save Changes**.
7. You will see an error message that tells you the Kudu keytab is missing. To generate the keytab, go to the top navigation bar and click **Administration** > **Security**.
8. Go to the **Kerberos Credentials** tab. On this page you will see a list of the existing Kerberos principals for services running on the cluster.
9. Click **Generate Missing Credentials**. Once the Generate Missing Credentials command has finished running, you will see the Kudu principal added to the list.

Configuring Coarse-grained Authorization with ACLs

1. Go to the **Kudu** service.

2. Click the **Configuration** tab.
3. Select **Category** > **Security**.
4. In the Search field, type **ACL** to show the relevant properties.
5. Edit the following properties according to your cluster configuration:

Field	Usage Notes
***Superuser* Access Control List**	Add a comma-separated list of superusers who can access the cluster. By default, this property is left blank. '*' indicates that all authenticated users will be given superuser access.
***User* Access Control List**	Add a comma-separated list of users who can access the cluster. By default, this property is set to '*'. The default value of `'*'` allows all users access to the cluster. However, if authentication is enabled, this will restrict access to only those users who are able to successfully authenticate using Kerberos. Unauthenticated users on the same network as the Kudu servers will be unable to access the cluster. Add the `impala` user to this list to allow Impala to query data in Kudu. You might choose to add any other relevant usernames if you want to give access to Spark Streaming jobs.

6. Click **Save Changes**.

Configuring HTTPS Encryption for the Kudu Master and Tablet Server Web UIs

Use the following steps to enable HTTPS for encrypted connections to the Kudu master and tablet server web UIs.

1. Go to the **Kudu** service.
2. Click the **Configuration** tab.
3. Select **Category** > **Security**.
4. In the Search field, type **TLS/SSL** to show the relevant properties.
5. Edit the following properties according to your cluster configuration:

Field	Usage Notes
Master TLS/SSL Server Private Key File (PEM Format)	Set to the path containing the Kudu master host's private key (PEM-format). This is used to enable TLS/SSL encryption (over HTTPS) for browser-based connections to the Kudu master web UI.
Tablet Server TLS/SSL Server Private Key File (PEM Format)	Set to the path containing the Kudu tablet server host's private key (PEM-format). This is used to enable TLS/SSL encryption (over HTTPS) for browser-based connections to Kudu tablet server web UIs.
Master TLS/SSL Server Certificate File (PEM Format)	Set to the path containing the signed certificate (PEM-format) for the Kudu master host's private key (set in Master TLS/SSL Server Private Key File). The certificate file can be created by concatenating all the appropriate root and intermediate certificates required to verify trust.
Tablet Server TLS/SSL Server Certificate File (PEM Format)	Set to the path containing the signed certificate (PEM-format) for the Kudu tablet server host's private key (set in Tablet Server TLS/SSL Server Private Key File). The certificate file can be created by concatenating all the appropriate root and intermediate certificates required to verify trust.
Enable TLS/SSL for Master Server	Enables HTTPS encryption on the Kudu master web UI.
Enable TLS/SSL for Tablet Server	Enables HTTPS encryption on the Kudu tablet server Web UIs.

6. Click **Save Changes**.

Configuring a Secure Kudu Cluster using the Command Line

Important: Follow these command-line instructions on systems that do not use Cloudera Manager. If you are using Cloudera Manager, see Configuring a Secure Kudu Cluster using Cloudera Manager on page 443.

The following configuration parameters should be set on all servers (master and tablet servers) to ensure that a Kudu cluster is secure:

```
# Connection Security
#--------------------
--rpc-authentication=required
--rpc-encryption=required
--keytab-file=<path-to-kerberos-keytab>

# Web UI Security
#--------------------
--webserver-certificate-file=<path-to-cert-pem>
--webserver-private-key-file=<path-to-key-pem>
# optional
--webserver-private-key-password-cmd=<password-cmd>

# If you prefer to disable the web UI entirely:
--webserver-enabled=false

# Coarse-grained authorization
#--------------------------------

# This example ACL setup allows the 'impala' user as well as the
# 'etl_service_account' principal access to all data in the
# Kudu cluster. The 'hadoopadmin' user is allowed to use administrative
# tooling. Note that by granting access to 'impala', other users
# may access data in Kudu via the Impala service subject to its own
# authorization rules.
--user-acl=impala,etl_service_account
--admin-acl=hadoopadmin
```

More information about these flags can be found in the configuration reference documentation.

Security How-To Guides

Configuring security for clusters can be complex and time-consuming. Here are some start-to-finish guides and single-focused instruction sets aimed at simplifying various tasks.

Administrative Basics

- Check Cluster Security Settings
- Configure Antivirus Software on CDH Hosts
- Log a Security Support Case

Cloud Security

Amazon Web Services (AWS) and Amazon S3

- Amazon Web Services (AWS) Security on page 447
- How to Configure AWS Credentials on page 471
- Configure Authentication for Amazon S3
- Configure Encryption for Amazon S3

Microsoft Azure

- How To Set Up Access to Cloudera EDH or Cloudera Director (Microsoft Azure Marketplace) on page 488

Client Access

- Configure Browsers for Kerberos Authentication (SPNEGO)
- Set Up a Gateway Node to Restrict Access to the Cluster

Data Privacy

- Enable Sensitive Data Redaction

Data in Transit Encryption

- Add Root and Intermediate CAs to Truststore for TLS/SSL
- How to Configure Encrypted Transport for HBase Data on page 454Configure Encrypted Transport for HBase Data
- How to Configure Encrypted Transport for HDFS Data on page 455Configure Encrypted Transport for HDFS Data
- Configure TLS Encryption for Cloudera Manager (Start-to-finish, Cloudera TLS Level 3)
- Convert DER, JKS, PEM for TLS/SSL Clients and Services
- Obtain and Deploy Keys and Certificates for TLS/SSL
- Use Self-Signed Certificates for TLS

How to Add Root and Intermediate CAs to Truststore for TLS/SSL

If a signed certificate is from a certificate authority (CA) that does not have certificates in the truststore for whatever reason (internal CA or a public CA not included in the Java truststore, for example), you must explicitly establish trust for the CA, as detailed below. The content of the truststore for the Cloudera Manager Server cluster can be modified (certificates added, for example) without restarting the server. Changes to the truststore are adopted by the system within 10 seconds.

Explicit Trust for Certificates

Before importing the certificate into the keystore of the host system, you must load the root CAs and any intermediate CAs into the truststore.

1. Copy the root and intermediate CA certificates to these locations on the Cloudera Manager Server host:

```
/opt/cloudera/security/pki/rootca.cert.pem
/opt/cloudera/security/pki/intca.cert.pem
```

   a. For concatenated files containing root CA and intermediate CA certificates, split the file between the `END CERTIFICATE` and `BEGIN CERTIFICATE` boundaries that separate each certificate in the file and make individual files instead.
   b. When extracting multiple intermediate CA certificates from a concatenated file, use unique file names such as `intca-1.cert.pem`, `intca-1.cert.pem`, and so on.

2. Import the root CA certificate into the JDK truststore. If you do not have the `$JAVA_HOME` variable set, replace it with the path to the Oracle JDK.

```
$ sudo keytool -importcert -alias rootca -keystore $JAVA_HOME/jre/lib/security/jssecacerts
 \
-file /opt/cloudera/security/pki/rootca.cert.pem -storepass changeit
```

   The default password for the `cacerts` file is `changeit` (as shown in the above command). Cloudera recommends changing this password by running the keytool command:

```
keytool -storepasswd -keystore $JAVA_HOME/jre/lib/security/cacerts
```

3. Copy the `jssecacerts` file from the Cloudera Manager Server host to all other cluster hosts. Copy the file to the same location on each host using the path required by Oracle JDK, which is as follows:

```
$JAVA_HOME/jre/lib/security/jssecacerts
```

4. On the Cloudera Manager Server host, append the intermediate CA certificate to the signed server certificate. Be sure to use the **append** (>>) operator—not overwrite (>)—when executing the statement:

```
$ sudo cat /opt/cloudera/security/pki/intca.cert.pem >> \
/opt/cloudera/security/pki/$(hostname -f)-server.cert.pem
```

Amazon Web Services (AWS) Security

Amazon Web Services (AWS) is Amazon's cloud solution that offers compute, storage, networking, and other infrastructure services that can be used for Cloudera cluster deployments, whether completely cloud-based or in combination with on-premises clusters.

For example, Amazon Elastic Compute Cloud (EC2) can be used for the instances that make-up the nodes of a Cloudera cluster deployed to the AWS cloud. Amazon's cloud-based storage solution, Amazon Simple Storage Service (Amazon S3), can be used by both on-premises and AWS-cloud-based clusters in various ways, including as storage for Impala tables for direct use by Hue and Hive, and other CDH components such as HDFS client, Hive, Impala, MapReduce.

As of release 5.11, Cloudera Manager supports Amazon's IAM-role based access to Amazon S3 storage, in addition to its prior support of AWS access key and secret key. See How to Configure AWS Credentials on page 471 for details.

For any AWS service, including Amazon S3, you must obtain an Amazon Web Services account and have appropriate access to the AWS Management Console to set up the various services you want, including Amazon S3. Assuming you have an account for AWS, to provide access from your Cloudera cluster to Amazon S3 storage you must configure AWS credentials.

- Configuring Authentication
- Configuring Encryption

Getting Started with Amazon Web Services

To get started with AWS, including Amazon S3, you must have:

1. An Amazon Web Services account. Both Amazon and Cloudera recommend that you do not use your primary Amazon account—known as the root account—for working with Amazon S3 and other AWS services. See the AWS IAM documentation for details about how to set up your AWS account.
2. Access to the AWS Management Console and appropriate permissions to create and configure the AWS services needed for your use case, such as the following:
   a. AWS Elastic Compute Cloud (EC2) to deploy your cluster to the AWS cloud.
   b. AWS Identity and Access Management (IAM) to set up users and groups, or to set up an IAM role.
   c. Amazon S3 and the specific storage bucket (or buckets) for use with your cluster.
   d. Amazon DynamoDB to enable the database needed by Cloudera S3Guard, if you plan to enable S3Guard for your cluster. Cloudera S3Guard augments Amazon S3 with a database to track metadata changes so that the 'eventual consistency' model inherent to Amazon S3 does not pose a problem for transactions or other use cases in which changes may not be apparent to each other in real time. See Configuring and Managing S3Guard in Cloudera Administration for details. To use S3Guard, you will also need to set up the appropriate access policy (create table, read, write) to DynamoDB for the same AWS identity that owns the Amazon S3 storage.
   e. AWS Key Management Services (KMS) (AWS KMS) to create encryption keys for your Amazon S3 bucket if you plan to use SSE-KMS for server-side encryption (not necessary for SSE-S3 encryption. See How to Configure Encryption for Amazon S3 for details).

Configuration Properties Reference

This table provides reference documentation for the `core-site.xml` properties relevant for use with AWS and Amazon S3.

Property	Description
fs.s3a.server-side-encryption-algorithm	Enable server-side encryption for the Amazon S3 storage bucket associated with the cluster. Allowable values: • **`AES256`** Specifies SSE-S3 server-side encryption for Amazon S3. • **`SSE-KMS`** Specifies SSE-KMS server-side encryption for Amazon S3. Requires adding the `fs.s3a.server-side-encryption-key` property with a valid value.
fs.s3a.server-side-encryption-key	Specify the ARN, ARN plus alias, Alias, or globally unique ID of the key created in AWS Key Management Service for use with SSE-KMS.
fs.s3a.awsAccessKeyId	Specify the AWS access key ID. This property is irrelevant and not used to access Amazon S3 storage from a cluster launched using an IAM role.
fs.s3a.awsSecretAccessKey	Specify the AWS secret key provided by Amazon. This property is irrelevant and not used to access Amazon S3 storage from a cluster launched using an IAM role.
fs.s3a.endpoint	Use this property only if the endpoint is outside the standard region (s3.amazonaws.com), such as regions and endpoints in China or in the US GovCloud. See AWS regions and endpoints documentation for details.
fs.s3a.connection.ssl.enabled	Enables (`true`) and disables (`false`) TLS/SSL connections to Amazon S3. Default is `true`.

How to Check Security Settings on a Cluster

Quickly perform a high level check of your cluster's security configuration by doing one of the following:

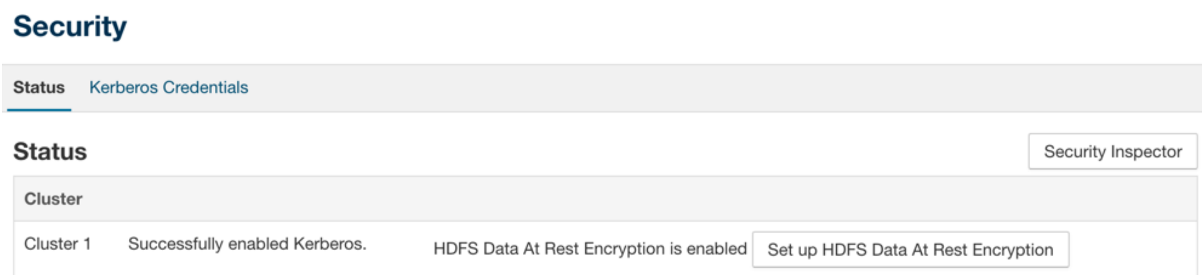
Check Security for Cloudera Manager Clusters

Use Cloudera Manager to verify security mechanisms for your cluster by simply examining the properties for the cluster.

For clusters not managed by Cloudera Manager Server, see Check Security for CDH Clusters on page 450 .

To check Kerberos and HDFS encryption:

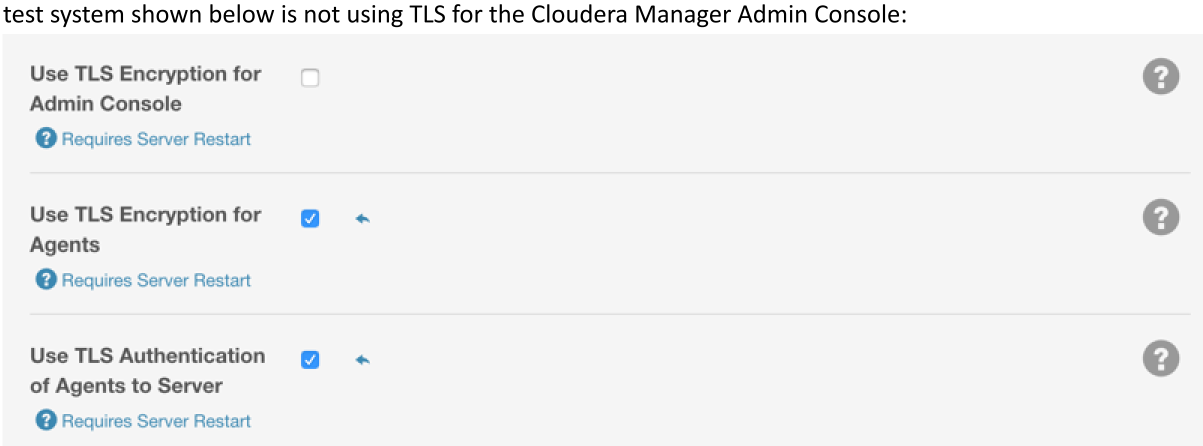
1. Log into the Cloudera Manager Admin Console.
2. Select **Security** from the **Administration** drop-down selector to display a list of managed clusters:



This shows at a glance that both Kerberos and HDFS transparent encryption have been configured for this cluster.

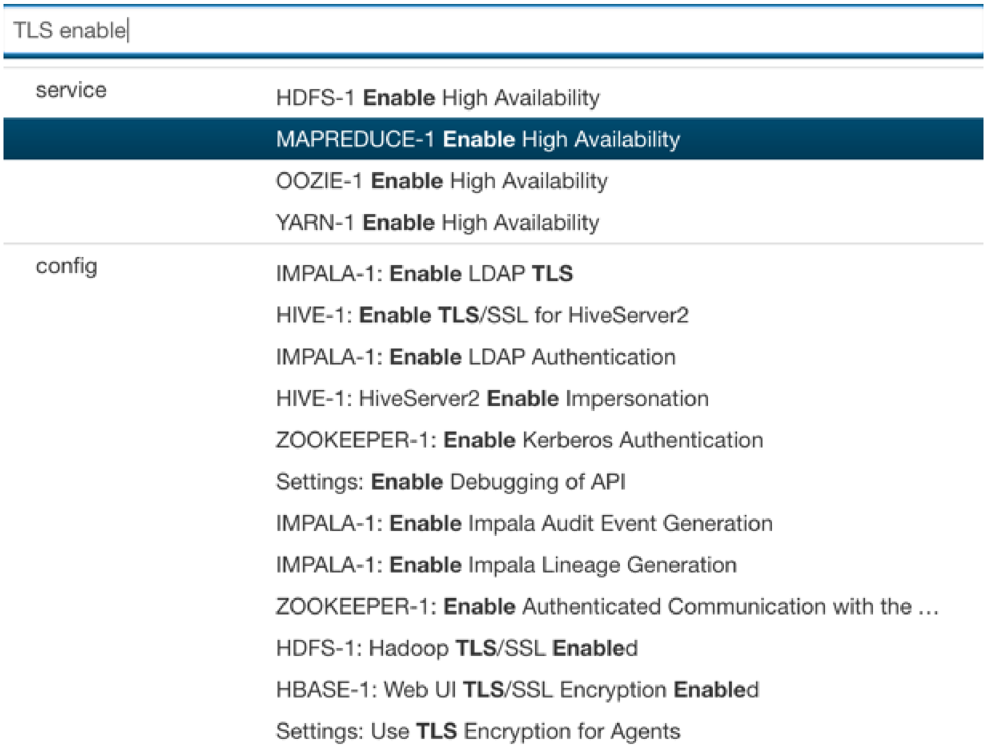
To check TLS settings:

1. Select **Settings** from the **Administration** drop-down selector to open a search field.
2. Enter **TLS** in the search field to display all TLS related configuration settings.
3. Scroll through the displayed results, looking for "Use TLS..." for various services and processes. For example, the test system shown below is not using TLS for the Cloudera Manager Admin Console:



See How to Configure TLS Encryption for Cloudera Manager for complete information about configuring TLS for the cluster.

To find all TLS settings, cluster-wide, enter **"TLS enabled"** (or simply, "TLS") in the top-most search field on the Cloudera Manager Admin Console. Then you can easily select from among the display list to examine the actual setting.



Check Security for CDH Clusters

To check security settings for CDH components not managed by Cloudera Manager, open the configuration file (`core-site.xml`) in a text editor and examine the property values shown below:

Functionality	Property	Value
TLS	hadoop.ssl.enabled	true
Kerberos	hadoop.security.authentication	kerberos
	hadoop.security.authorization	true

See Configuring Authentication in CDH Using the Command Line and Configuring TLS/SSL Encryption for CDH Services for more information.

How to Use Antivirus Software on CDH Hosts

If you use antivirus software on your servers, consider configuring it to skip scans on certain types of Hadoop-specific resources. It can take a long time to scan large files or directories with a large number of files. In addition, if your antivirus software locks files or directories as it scans them, those resources will be unavailable to your Hadoop processes during the scan, and can cause latency or unavailability of resources in your cluster. Consider skipping scans on the following types of resources:

- Scratch directories used by services such as Impala
- Log directories used by various Hadoop services
- Data directories which can grow to petabytes in size

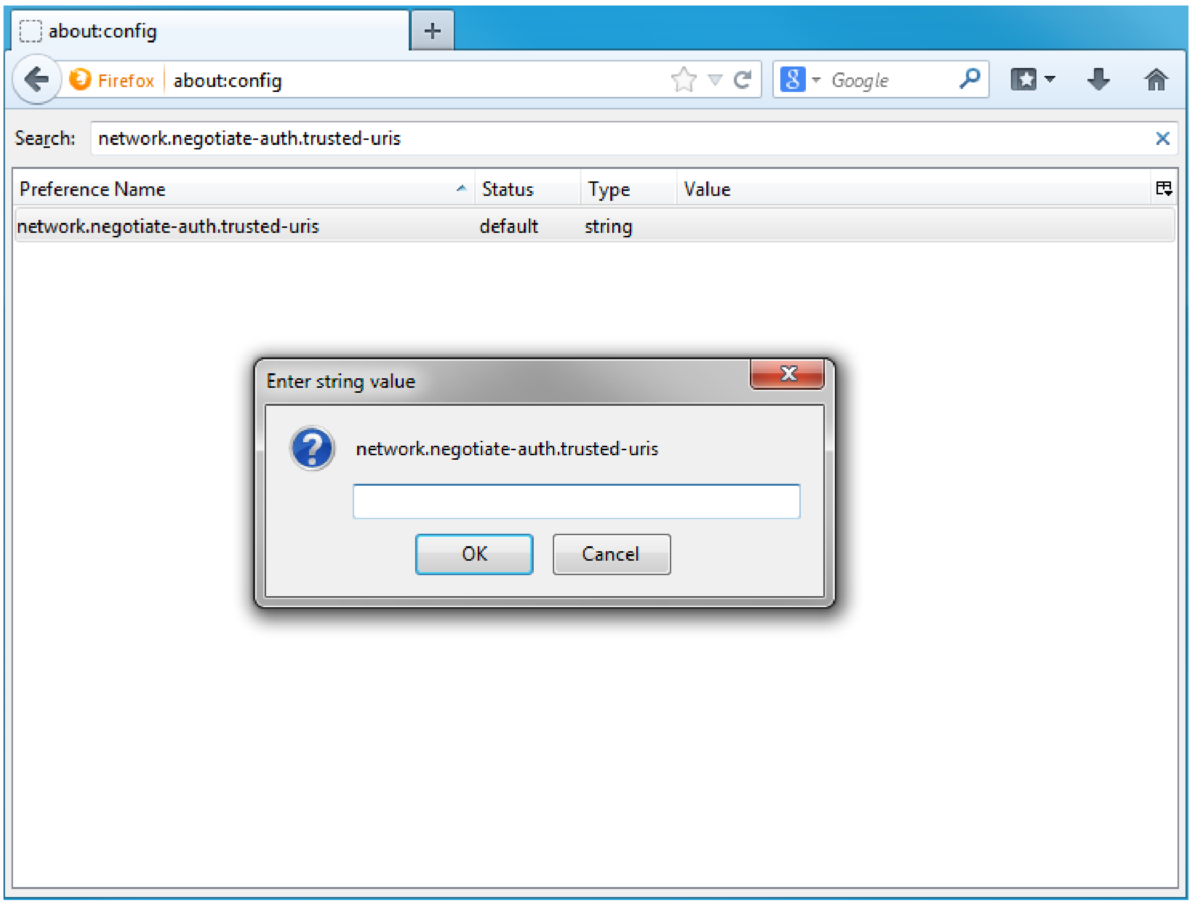
The specific directory names and locations depend on the services your cluster uses and your configuration. In general, avoid scanning very large directories and filesystems. Instead, limit write access to these locations using security mechanisms such as access controls at the level of the operating system, HDFS, or at the service level.

How to Configure Browsers for Kerberos Authentication

The browser configurations below are required only for those browsers used to connect to Cloudera Manager instances with the Require Authentication for HTTP Web Consoles configuration property enabled. The settings below enable the respective browser to use SPNEGO to negotiate Kerberos authentication for the browser.

Mozilla Firefox

1. Open the low level Firefox configuration page by loading the `about:config` page.
2. In the **Search** text box, enter: `network.negotiate-auth.trusted-uris`
3. Double-click the `network.negotiate-auth.trusted-uris` preference and enter the hostname or the domain of the web server that is protected by Kerberos HTTP SPNEGO. Separate multiple domains and hostnames with a comma.
4. Click **OK**.



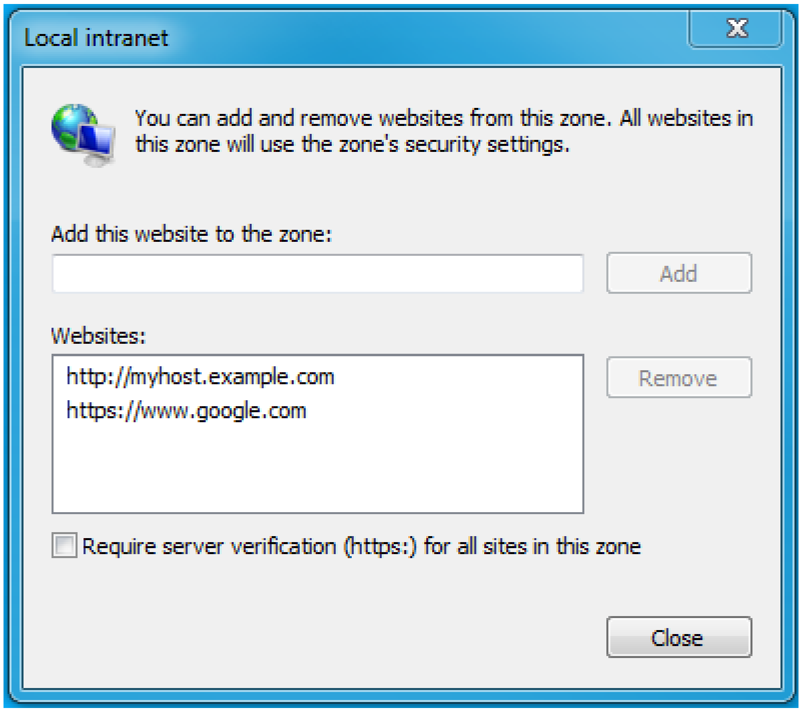
Internet Explorer

Follow the steps below to configure Internet Explorer.

Configuring the Local Intranet Domain

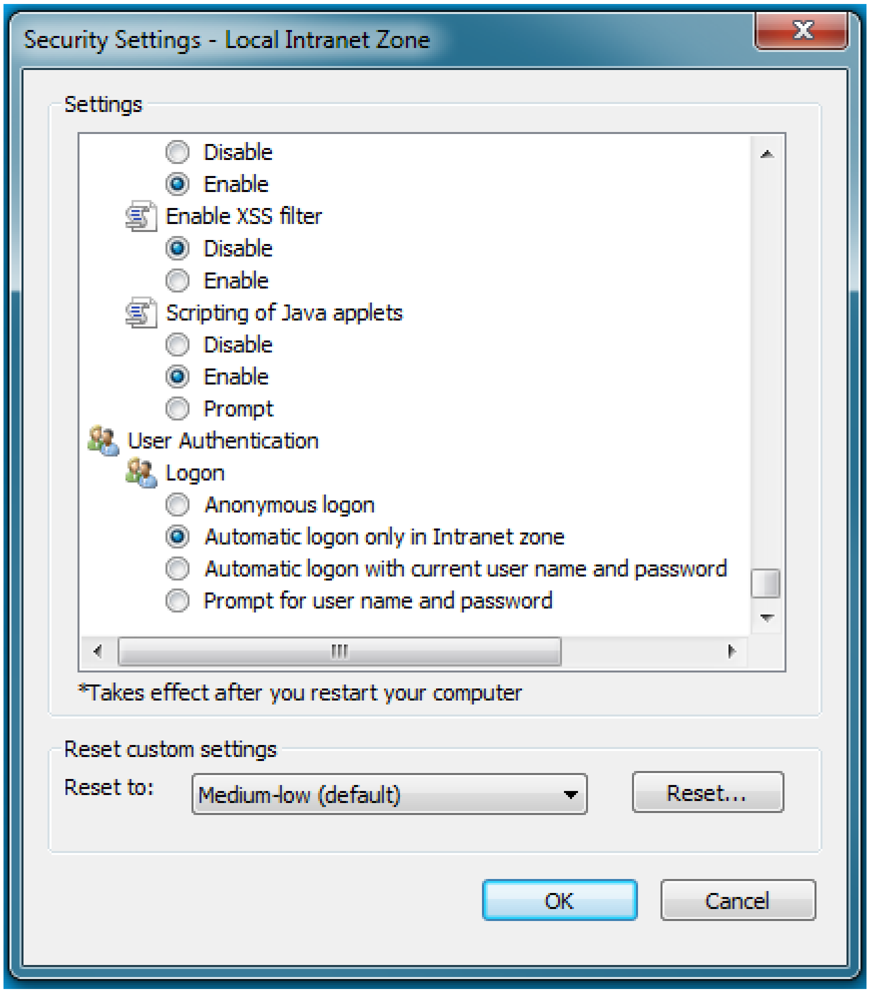
1. Open Internet Explorer and click the Settings gear icon in the top-right corner. Select **Internet options**.
2. Select the **Security** tab.
3. Select the **Local Intranet** zone and click the **Sites** button.
4. Make sure that the first two options, **Include all local (intranet) sites not listed in other zones** and **Include all sites that bypass the proxy server** are checked.

5. Click **Advanced** and add the names of the domains that are protected by Kerberos HTTP SPNEGO, one at a time, to the list of websites. For example, `myhost.example.com`. Click **Close**.
6. Click **OK** to save your configuration changes.



Configuring Intranet Authentication

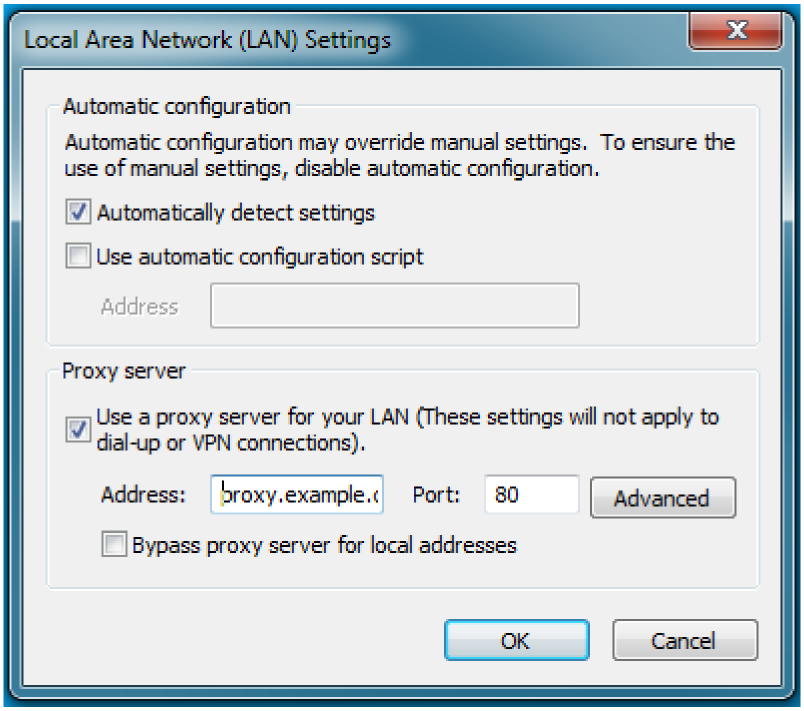
1. Click the Settings gear icon in the top-right corner. Select **Internet options**.
2. Select the **Security** tab.
3. Select the **Local Intranet** zone and click the **Custom level...** button to open the **Security Settings - Local Intranet Zone** dialog box.
4. Scroll down to the **User Authentication** options and select **Automatic logon only in Intranet zone**.
5. Click **OK** to save these changes.



Verifying Proxy Settings

these steps only if you have a proxy server already enabled.

1. Click the Settings gear icon in the top-right corner. Select **Internet options**.
2. Select the **Connections** tab and click **LAN Settings**.
3. Verify that the proxy server **Address** and **Port** number settings are correct.
4. Click **Advanced** to open the **Proxy Settings** dialog box.
5. Add the Kerberos-protected domains to the **Exceptions** field.
6. Click **OK** to save any changes.



Google Chrome

For Windows:

- Open the Control Panel to access the **Internet Options** dialog. Use the same configuration as detailed in Configuration changes required are the same as those described above for Internet Explorer.

For Linux or MacOS:

- Add the `--auth-server-whitelist` parameter to the `google-chrome` command. For example, to run Chrome from a Linux prompt, run the `google-chrome` command as follows:

```
> google-chrome --auth-server-whitelist = "hostname/domain"
```

How to Configure Encrypted Transport for HBase Data

This topic describes how to configure encrypted HBase data transport using Cloudera Manager and the command line.

Configuring Encrypted HBase Data Transport Using Cloudera Manager

Minimum Required Role: Full Administrator

To enable encryption of data transferred between HBase masters and RegionServers and between RegionServers and clients:

1. Enable Hadoop security using Kerberos.
2. Configure Kerberos authentication for HBase.
3. Select the HBase service.
4. Click the **Configuration** tab.
5. Select **Scope** > **HBase (Service Wide)**.
6. Select **Category** > **Security**.
7. Search for the HBase Transport Security property and select one of the following:
    - `authentication`: Enables simple authentication using Kerberos.

- `integrity`: Checks the integrity of data received to ensure it was not corrupted in transit. Selecting `integrity` also enables `authentication`.
- `privacy`: Ensures privacy by encrypting the data in transit using TLS/SSL encryption. Selecting `privacy` also enables `authentication` and `integrity`.

  Set this property to **privacy** to enable secure RPC transport.

8. Click **Save Changes**.
9. Restart the HDFS service.

Configuring Encrypted HBase Data Transport Using the Command Line

> **Important:**
>
> - This configuration process can be completed using either Cloudera Manager or the command-line instructions.
> - This information applies specifically to CDH 5.13.x. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

1. Enable Hadoop Security using Kerberos.
2. Enable HBase security using Kerberos.
3. Enable RPC encryption by setting `hbase.rpc.protection` in the `hbase-site.xml` file to one of the following:
   - `authentication`: Enables simple authentication using Kerberos.
   - `integrity`: Checks the integrity of data received to ensure it was not corrupted in transit. Selecting `integrity` also enables `authentication`.
   - `privacy`: Ensures privacy by encrypting the data in transit using TLS/SSL encryption. Selecting `privacy` also enables `authentication` and `integrity`.

     Set this property to **privacy** to enable secure RPC transport.
4. Restart all daemons.

How to Configure Encrypted Transport for HDFS Data

This topic describes how to configure encrypted HDFS data transport using both, Cloudera Manager, and the command line.

You must enable Kerberos before configuring encrypted HDFS data transport. See Authentication on page 40 for instructions.

Using Cloudera Manager

Minimum Required Role: Full Administrator

To enable encryption of data transferred between DataNodes and clients, and among DataNodes, proceed as follows:

1. Enable Hadoop security using Kerberos.
2. Select the HDFS service.
3. Click the **Configuration** tab.
4. Select **Scope** > **HDFS (Service Wide)**
5. Select **Category** > **Security**.
6. Configure the following properties: (You can type the property name in the **Search** box to locate the property.)

Property	Description
Enable Data Transfer Encryption	Check this field to enable wire encryption.
Data Transfer Encryption Algorithm	Optionally configure the algorithm used to encrypt data.
Hadoop RPC Protection	Select **privacy**.

7. Click **Save Changes**.
8. Restart the HDFS service.

Using the Command Line

Important:

- This configuration process can be completed using either Cloudera Manager or the command-line instructions.
- This information applies specifically to CDH 5.13.x. If you use an earlier version of CDH, see the documentation for that version located at Cloudera Documentation.

To enable encrypted data transport using the command line, proceed as follows:

1. Enable Kerberos authentication, following these instructions.
2. Set the optional RPC encryption by setting `hadoop.rpc.protection` to "`privacy`" in the `core-site.xml` file in both client and server configurations.

   Note:

   If RPC encryption is not enabled, transmission of other HDFS data is also insecure.

3. Set `dfs.encrypt.data.transfer` to `true` in the `hdfs-site.xml` file on all server systems.
4. Restart all daemons.

How to Configure TLS Encryption for Cloudera Manager

When you configure authentication and authorization on a cluster, Cloudera Manager Server sends sensitive information over the network to cluster hosts, such as Kerberos keytabs and configuration files that contain passwords. To secure this transfer, you must configure TLS encryption between Cloudera Manager Server and all cluster hosts.

TLS encryption is also used to secure client connections to the Cloudera Manager Admin Interface, using HTTPS.

Cloudera Manager also supports TLS authentication. Without certificate authentication, a malicious user can add a host to Cloudera Manager by installing the Cloudera Manager Agent software and configuring it to communicate with Cloudera Manager Server. To prevent this, you must install certificates on each agent host and configure Cloudera Manager Server to trust those certificates.

This guide shows how to configure and enable TLS encryption and certificate authentication for Cloudera Manager. The provided examples use an internal certificate authority (CA) to sign all TLS certificates, so this guide also shows you how to establish trust with the CA. (For certificates signed by a trusted public CA, establishing trust is not necessary, because the Java Development Kit (JDK) already trusts them.)

Use this guide to enable TLS encryption and certificate authentication for Cloudera Manager:

Generate TLS Certificates

> **Important:**
>
> - You must use the Oracle JDK `keytool` utility. Do not use other JDK (such as OpenJDK) command line tools for this procedure. If you have multiple JDKs, set the PATH variable such that the Oracle JDK is first. For example:
>
> ```
> $ export JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera
> $ export PATH=$JAVA_HOME/bin:$PATH
> ```
>
> - Use the same password for the `-keypass` and `-storepass` values. Cloudera Manager does not support using different passwords for the key and keystore.

Before configuring Cloudera Manager Server and all Cloudera Manager Agents to use TLS encryption, generate the server and agent certificates:

Generate the Cloudera Manager Server Certificate

The following procedure assumes that an internal certificate authority (CA) is used, and shows how to establish trust for that internal CA. If you are using a trusted public CA (such as Symantec, GeoTrust, Comodo, and others), you do not need to explicitly establish trust for the issued certificates, unless you are using an older JDK and a newer public CA. Older JDKs might not trust newer public CAs by default.

1. On the Cloudera Manager Server host, create the `/opt/cloudera/security/pki` directory:

```
$ sudo mkdir -p /opt/cloudera/security/pki
```

   If you choose to use a different directory, make sure you use the same directory on all cluster hosts to simplify management and maintenance.

2. On the Cloudera Manager Server host, use the `keytool` utility to generate a Java keystore and certificate signing request (CSR). Replace the `OU`, `O`, `L`, `ST`, and `C` entries with the values for your environment. Use the same password for the `-keypass` and `-storepass` values. Cloudera Manager does not support using different passwords for the key and keystore.

```
$ sudo keytool -genkeypair -alias $(hostname -f)-server -keyalg RSA -keystore \
/opt/cloudera/security/pki/$(hostname -f)-server.jks -keysize 2048 -dname \
"CN=$(hostname -f),OU=Engineering,O=Cloudera,L=Palo Alto,ST=California,C=US" \
-storepass password -keypass password
$ sudo keytool -certreq -alias $(hostname -f)-server \
-keystore /opt/cloudera/security/pki/$(hostname -f)-server.jks \
-file /opt/cloudera/security/pki/$(hostname -f)-server.csr -storepass password \
-keypass password
```

3. Submit the CSR file (for example, `cm01.example.com-server.csr`) to your certificate authority to obtain a server certificate. If possible, obtain the certificate in PEM (Base64 ASCII) format. The certificate file is in PEM format if it looks like this:

```
-----BEGIN CERTIFICATE-----
<The encoded certificate is represented by multiple lines of exactly 64 characters,
except
for the last line, which can contain 64 characters or fewer.>
-----END CERTIFICATE-----
```

   If your issued certificate is in binary (DER) format, convert it to PEM format.

4. After you receive the signed certificate, copy it to `/opt/cloudera/security/pki/$(hostname -f)-server.cert.pem`.
5. Copy the root and intermediate CA certificates to `/opt/cloudera/security/pki/rootca.cert.pem` and `/opt/cloudera/security/pki/intca.cert.pem` on the Cloudera Manager Server host. If you have a concatenated file containing the root CA and an intermediate CA certificate, split the file along the END

CERTIFICATE/BEGIN CERTIFICATE boundary into individual files. If there are multiple intermediate CA certificates, use unique file names such as intca-1.cert.pem, intca-1.cert.pem, and so on.

6. On the Cloudera Manager Server host, copy the JDK cacerts file to jssecacerts:

```
$ sudo cp $JAVA_HOME/jre/lib/security/cacerts $JAVA_HOME/jre/lib/security/jssecacerts
```

If you do not have the $JAVA_HOME variable set, replace it with the path to the Oracle JDK (for example, /usr/java/jdk1.7.0_67-cloudera/).

> **Note:** The Oracle JDK uses the jssecacerts file for its default truststore if it exists. Otherwise, it uses the cacerts file. Creating the jssecacerts file allows you to trust an internal CA without modifying the cacerts file that is included with the JDK.

7. Import the root CA certificate into the JDK truststore. If you do not have the $JAVA_HOME variable set, replace it with the path to the Oracle JDK.

```
$ sudo keytool -importcert -alias rootca -keystore $JAVA_HOME/jre/lib/security/jssecacerts
 \
-file /opt/cloudera/security/pki/rootca.cert.pem -storepass changeit
```

The default password for the cacerts file is changeit. Cloudera recommends changing this password by running the command keytool -storepasswd -keystore $JAVA_HOME/jre/lib/security/cacerts.

8. Copy the jssecacerts file from the Cloudera Manager Server host to all other cluster hosts. Make sure you copy the file to the correct location ($JAVA_HOME/jre/lib/security/jssecacerts), because the Oracle JDK expects it there.

9. On the Cloudera Manager Server host, append the intermediate CA certificate to the signed server certificate, and then import it into the keystore. Make sure that you use the append operator (>>) and not the overwrite operator (>):

```
$ sudo cat /opt/cloudera/security/pki/intca.cert.pem >>
/opt/cloudera/security/pki/$(hostname -f)-server.cert.pem
$ sudo keytool -importcert -alias $(hostname -f)-server \
-file /opt/cloudera/security/pki/$(hostname -f)-server.cert.pem \
-keystore /opt/cloudera/security/pki/$(hostname -f)-server.jks
```

If you see a message like the following, enter yes to continue:

```
... is not trusted. Install reply anyway? [no]:  yes
```

You *must* see the following response verifying that the certificate has been properly imported against its private key.

```
Certificate reply was installed in keystore
```

If you do not see this response, contact Cloudera Support.

Generate the Cloudera Manager Agent Certificates

Complete the following procedure on each Cloudera Manager Agent host. The provided examples continue to use an internal certificate authority (CA) to sign the agent certificates.

1. On all Cloudera Manager Agent hosts, create the /opt/cloudera/security/pki directory:

```
$ sudo mkdir -p /opt/cloudera/security/pki
```

If you choose to use a different directory, make sure you use the same directory on all cluster hosts to simplify management and maintenance.

2. On all Cloudera Manager Agent hosts, create a Java Keystore and private key as follows:

```
$ keytool -genkeypair -alias $(hostname -f)-agent -keyalg RSA -keystore \
/opt/cloudera/security/pki/$(hostname -f)-agent.jks -keysize 2048 -dname \
"CN=$(hostname -f),OU=Engineering,O=Cloudera,L=Palo Alto,ST=California,C=US" \
-storepass password -keypass password
```

   Use the same password for the `-keypass` and `-storepass` values. Cloudera Manager does not support using different passwords for the key and keystore.

3. On all Cloudera Manager Agent hosts, generate the certificate signing request (CSR) and submit it to a CA. Use the `keytool` extended attributes to specify both `serverAuth` and `clientAuth` options:

```
$ keytool -certreq -alias $(hostname -f)-agent  \
-keystore /opt/cloudera/security/pki/$(hostname -f)-agent.jks  \
-file /opt/cloudera/security/pki/$(hostname -f)-agent.csr  \
-ext EKU=serverAuth,clientAuth  \
-storepass password -keypass password
```

   For security purposes, many commercial CAs ignore requested extensions in a CSR. Make sure that you inform the CA that you require certificates with both server and client authentication options.

4. For each signed certificate you receive, copy it to `/opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem` on the correct host.
5. Inspect the certificates to verify that both server and client authentication options are present:

```
$ openssl x509 -in /opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem -noout -text
```

   Look for output similar to the following:

```
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
```

   If the certificate does not have both `TLS Web Server Authentication` and `TLS Web Client Authentication` listed in the `X509v3 Extended Key Usage` section, re-submit the CSR to the CA, and request that they generate a certificate that can be used for both server and client authentication.

6. Copy the root and intermediate CA certificates to `/opt/cloudera/security/pki/rootca.cert.pem` and `/opt/cloudera/security/pki/intca.cert.pem` on each Cloudera Manager Agent host. If you have a concatenated file containing the root CA and an intermediate CA certificate, split the file along the `END CERTIFICATE`/`BEGIN CERTIFICATE` boundary into individual files. If there are multiple intermediate CA certificates, use unique file names such as `intca-1.cert.pem`, `intca-1.cert.pem`, and so on.
7. On each Cloudera Manager Agent host, append the intermediate CA certificate to the signed certificate, and then import it into the keystore. Make sure that you use the append operator (>>) and not the overwrite operator (>):

```
$ sudo cat /opt/cloudera/security/pki/intca.cert.pem >>
/opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem
$ sudo keytool -importcert -alias $(hostname -f)-agent \
-file /opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem \
-keystore /opt/cloudera/security/pki/$(hostname -f)-agent.jks
```

   If you see a message like the following, enter `yes` to continue:

```
... is not trusted. Install reply anyway? [no]:  yes
```

   You *must* see the following response verifying that the certificate has been properly imported against its private key.

```
Certificate reply was installed in keystore
```

   If you do not see this response, [contact Cloudera Support](#).

8. On each Cloudera Manager Agent host, create symbolic links (symlink) for the certificate and keystore files:

```
$ ln -s /opt/cloudera/security/pki/$(hostname -f)-agent.cert.pem
/opt/cloudera/security/pki/agent.cert.pem
$ ln -s /opt/cloudera/security/pki/$(hostname -f)-agent.jks
/opt/cloudera/security/pki/agent.jks
```

This allows you to use the same `/etc/cloudera-scm-agent/config.ini` file on all agent hosts rather than maintaining a file for each agent.

Configuring TLS Encryption for the Cloudera Manager Admin Console

Minimum Required Role: **Cluster Administrator** (also provided by **Full Administrator**)

Use the following procedure to enable TLS encryption for the Cloudera Manager Server admin interface. Make sure you have generated the server certificate as described in Generate the Cloudera Manager Server Certificate on page 457.

Step 1: Enable HTTPS for the Cloudera Manager Admin Console

1. Log in to the Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Select the **Security** category.
4. Configure the following TLS settings:

Property	Description
Cloudera Manager TLS/SSL Server JKS Keystore File Location	The complete path to the keystore file. In this example, the path is `/opt/cloudera/security/pki/`*`cm01.example.com`*`-server.jks`. Replace *cm01.example.com* with the Cloudera Manager Server hostname.
Cloudera Manager TLS/SSL Server JKS Keystore File Password	The password for the `/opt/cloudera/security/jks/`*`cm01.example.com`*`-server.jks` keystore.
Use TLS Encryption for Admin Console	Check this box to enable TLS encryption for Cloudera Manager.

5. Click **Save Changes** to save the settings.

Step 2: Specify SSL Truststore Properties for Cloudera Management Services

When enabling TLS for the Cloudera Manager Server admin interface, you must set the Java truststore location and password in the Cloudera Management Services configuration. Otherwise, roles such as Host Monitor and Service Monitor cannot connect to Cloudera Manager Server and will not start.

Configure the path and password for the `$JAVA_HOME/jre/lib/security/jssecacerts` truststore that you created earlier. Make sure that you copied this file to all cluster hosts, including the Cloudera Management Service hosts.

1. Open the Cloudera Manager Administration Console and go to the **Cloudera Management Service** service.
2. Click the **Configuration** tab.
3. Select **Scope** > **Cloudera Management Service (Service-Wide)**.
4. Select **Category** > **Security**.
5. Edit the following TLS/SSL properties according to your cluster configuration.

Property	Description
TLS/SSL Client Truststore File Location	The path to the client truststore file used in HTTPS communication. This truststore contains certificates of trusted servers, or of Certificate Authorities trusted to identify servers. For this example, set the value to

Property	Description
	`$JAVA_HOME/jre/lib/security/jssecacerts`. Replace `$JAVA_HOME` with the path to the Oracle JDK.
TLS/SSL Client Truststore File Password	The password for the truststore file.

6. Click **Save Changes** to commit the changes.

Step 3: Restart Cloudera Manager and Services

You must restart both Cloudera Manager Server and the Cloudera Management Service for TLS encryption to work. Otherwise, the Cloudera Management Services (such as Host Monitor and Service Monitor) cannot communicate with Cloudera Manager Server.

1. Restart the Cloudera Manager Server by running `service cloudera-scm-server restart` on the Cloudera Manager Server host.
2. After the restart completes, connect to the Cloudera Manager Admin Console using the HTTPS URL (for example: `https://`*`cm01.example.com`*`:7183`). If you used an internal CA-signed certificate, you must configure your browser to trust the certificate. Otherwise, you will see a warning in your browser any time you access the Cloudera Manager Administration Console. By default, certificates issued by public commercial CAs are trusted by most browsers, and no additional configuration is necessary if your certificate is signed by one of them.
3. Restart the Cloudera Management Service (**Cloudera Management Service** > **Actions** > **Restart**).

Configuring TLS Encryption for Cloudera Manager Agents

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

Use the following procedure to encrypt the communication between Cloudera Manager Server and Cloudera Manager Agents:

Step 1: Enable TLS Encryption for Agents in Cloudera Manager

Configure the TLS properties for Cloudera Manager Agents.

1. Log in to the Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Select the **Security** category.
4. Select the **Use TLS Encryption for Agents** option.
5. Click Save Changes.

Step 2: Enable TLS on Cloudera Manager Agent Hosts

To enable TLS between the Cloudera Manager agents and Cloudera Manager, you must specify values for the TLS properties in the `/etc/cloudera-scm-agent/config.ini` configuration file on all agent hosts.

1. On each agent host, open the `/etc/cloudera-scm-agent/config.ini` configuration file and set the `use_tls` parameter in the `[Security]` section as follows:

```
use_tls=1
```

Alternatively, you can edit the `config.ini` file on one host, and then copy it to the other hosts because this file by default does not contain host-specific information. If you have modified properties such as `listening_hostname` or `listening_ip address` in `config.ini`, you must edit the file individually on each host.

Step 3: Restart Cloudera Manager Server and Agents

Restart the Cloudera Manager Server with the following command to activate the TLS configuration settings.

```
$ sudo service cloudera-scm-server restart
```

On each agent host, restart the Cloudera Manager agent service:

```
$ sudo service cloudera-scm-agent restart
```

Step 4: Verify that the Cloudera Manager Server and Agents are Communicating

In the Cloudera Manager Admin Console, go to **Hosts** > **All Hosts**. If you see successful heartbeats reported in the **Last Heartbeat** column after restarting the agents, TLS encryption is working properly.

Enabling Server Certificate Verification on Cloudera Manager Agents

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

If you have completed the previous sections, communication between Cloudera Manager server and the agents is encrypted, but the certificate authenticity is not verified. For full security, you must configure the agents to verify the Cloudera Manager server certificate. If you are using a server certificate signed by an internal certificate authority (CA), you must configure the agents to trust that CA:

1. On each agent host, open the `/etc/cloudera-scm-agent/config.ini` configuration file, and then uncomment and set the following property:

```
verify_cert_file=/opt/cloudera/security/pki/rootca.cert.pem
```

   Alternatively, you can edit the `config.ini` file on one host, and then copy it to the other hosts because this file by default does not contain host-specific information. If you have modified properties such as `listening_hostname` or `listening_ip address` in `config.ini`, you must edit the file individually on each host.

2. Restart the Cloudera Manager agents. On each agent host, run the following command:

```
$ sudo service cloudera-scm-agent restart
```

3. Restart the Cloudera Management Service. On the **Home** > **Status** tab, click

   to the right of the Cloudera Management Service and select **Restart**.
4. Verify that the Cloudera Manager server and agents are communicating. In the Cloudera Manager Admin Console, go to **Hosts** > **All Hosts**. If you see successful heartbeats reported in the **Last Heartbeat** column after restarting the agents and management service, TLS verification is working properly. If not, check the agent log (`/var/log/cloudera-scm-agent/cloudera-scm-agent.log`) for errors.

Configuring Agent Certificate Authentication

> **Important:** Repeat this procedure on each agent host.

Without certificate authentication, a malicious user can add a host to Cloudera Manager by installing the Cloudera Manager agent software and configuring it to communicate with Cloudera Manager Server. To prevent this, you must configure Cloudera Manager to trust the agent certificates.

Step 1: Export the Private Key to a File

On each Cloudera Manager Agent host, use the `keytool` utility to export the private key and certificate to a PKCS12 file, which can then be split up into individual key and certificate files using the `openssl` command:

1. Export the private key and certificate:

```
$ keytool -importkeystore -srckeystore /opt/cloudera/security/pki/$(hostname -f)-agent.jks
 \
-srcstorepass password -srckeypass password -destkeystore
/opt/cloudera/security/pki/$(hostname -f)-agent.p12 \
-deststoretype PKCS12 -srcalias $(hostname -f)-agent -deststorepass password -destkeypass
 password
```

2. Use the `openssl` command to export the private key into its own file:

```
$ openssl pkcs12 -in /opt/cloudera/security/pki/$(hostname -f)-agent.p12 -passin
pass:password -nocerts \
-out /opt/cloudera/security/pki/$(hostname -f)-agent.key -passout pass:password
```

3. Create a symbolic link for the `.key` file:

```
$ ln -s /opt/cloudera/security/pki/$(hostname -f)-agent.key
/opt/cloudera/security/pki/agent.key
```

   This allows you to use the same `/etc/cloudera-scm-agent/config.ini` file on all agent hosts rather than maintaining a file for each agent.

Step 2: Create a Password File

The Cloudera Manager agent obtains the password from a text file, not from a command line parameter or environment variable. The password file allows you to use file permissions to protect the password. For example, run the following commands on each Cloudera Manager Agent host, or run them on one host and copy the file to the other hosts:

```
$ echo "password" > /etc/cloudera-scm-agent/agentkey.pw
$ sudo chown root:root /etc/cloudera-scm-agent/agentkey.pw
$ sudo chmod 440 /etc/cloudera-scm-agent/agentkey.pw
```

Replace *password* with the password you created in Step 1: Export the Private Key to a File on page 463.

Step 3: Configure the Agent to Use Private Keys and Certificates

On a Cloudera Manager Agent, open the `/etc/cloudera-scm-agent/config.ini` configuration file and edit the following properties.

Property	Example Value	Description
`client_key_file`	/opt/cloudera/security/pki/agent.key	Path to the private key file.
`client_keypw_file`	/etc/cloudera-scm-agent/agentkey.pw	Path to the private key password file.
`client_cert_file`	/opt/cloudera/security/pki/agent.cert.pem	Path to the client certificate file.

Copy the file to all other cluster hosts. If you have modified properties such as `listening_hostname` or `listening_ip address` in `config.ini`, you must edit the file individually on each host.

Step 4: Enable Agent Certificate Authentication

1. Log in to the Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Click the **Security** category.
4. Configure the following TLS settings:

Setting	Description
Use TLS Authentication of Agents to Server	Select this option to enable TLS authentication of agents to the server.
Cloudera Manager TLS/SSL Certificate Trust Store File	Specify the full filesystem path to the `jssecacerts` file located on the Cloudera Manager Server host. For example, `/usr/java/jdk1.7.0_67-cloudera/jre/lib/security/jssecacerts`.
Cloudera Manager TLS/SSL Certificate Trust Store Password	Specify the password for the `jssecacerts` truststore.

5. Click **Save Changes** to save the settings.

Step 5: Restart Cloudera Manager Server and Agents

1. On the Cloudera Manager server host, restart the Cloudera Manager server:

```
$ sudo service cloudera-scm-server restart
```

2. On every agent host, restart the Cloudera Manager agent:

```
$ sudo service cloudera-scm-agent restart
```

Step 6: Verify that Cloudera Manager Server and Agents are Communicating

In the Cloudera Manager Admin Console, go to **Hosts** > **All Hosts**. If you see successful heartbeats reported in the **Last Heartbeat** column after restarting the agents and server, TLS certificate authentication is working properly. If not, check the agent log (`/var/log/cloudera-scm-agent/cloudera-scm-agent.log`) for errors.

How to Convert File Encodings (DER, JKS, PEM) for TLS/SSL Certificates and Keys

Client and server processes require specific file formats for certificates, keys, and other digital artifacts used for TLS/SSL encryption. For example, when configured for TLS Level 2, Cloudera Manager Server presents Java KeyStore (JKS) formatted key and certificate to requesting Cloudera Manager Agent hosts. The Hue client also connects to Cloudera Manager Server, but Hue requires a PEM-formatted key and certificate, rather than JKS. The PEM format used by Cloudera Manager is PKCS #8, which handles certificates and keys as individual Base64-encoded text files.

If you receive binary DER files from your certificate authority, you must convert them to the appropriate format. Since neither Java Keytool nor OpenSSL work directly with PKCS format, many of the configuration tasks detailed in How to Configure TLS Encryption for Cloudera Manager on page 456 and Configuring Cloudera Manager Clusters for TLS/SSL on page 183 involve converting formats, or extracting keys or certificates from an artifact in one format to another.

Certificates issued by a CA in one format (encoding) can be used to create certificates in a different format using Java Keytool and OpenSSL as detailed below:

- Converting DER Encoded Certificates to PEM on page 464
- Converting JKS Key and Certificate to PEM on page 465
- Extracting the Private Key from PKCS Keystore on page 465
- Converting PEM Key and Certificate to JKS on page 465

Converting DER Encoded Certificates to PEM

OpenSSL can be used to convert a DER-encoded certificate to an ASCII (Base64) encoded certificate. Typically, DER-encoded certificates may have file extension of .DER, .CRT, or .CER, but regardless of the extension, a DER encoded certificate is not readable as plain text (unlike PEM encoded certificate).

A PEM-encoded certificate may also have file extension of .CRT or .CER, in which case, you can simply copy the file to a new name using the .PEM extension:

```
$ cp hostname.cer hostname.pem
```

To convert a DER-encoded certificate to PEM encoding using OpenSSL:

```
$ openssl x509 -inform der -in hostname.cer -out hostname.pem
```

For example:

```
$ openssl x509 -inform der -in /opt/cloudera/security/pki/hostname.cer -out
/tmp/hostname.pem
```

Converting JKS Key and Certificate to PEM

This process uses both Java keytool and OpenSSL (`keytool` and `openssl`, respectively, in the commands below) to export the composite private key and certificate from a Java keystore and then extract each element into its own file.

The `PKCS12` file created below is an interim file used to obtain the individual key and certificate files.

Replace *hostname-keystore*, *cmhost*, *hostname*, and *password* with values from your system.

1. Export the private key and certificate command line:

```
$ keytool -importkeystore -srckeystore /opt/cloudera/security/jks/hostname-keystore.jks
 \
-srcstorepass password -srckeypass password -destkeystore /tmp/hostname-keystore.p12 \
-deststoretype PKCS12 -srcalias hostname -deststorepass password -destkeypass password
```

2. Extract the certificate file from the resulting PKCS12 file:

```
$ openssl pkcs12 -in /tmp/hostname-keystore.p12 -passin pass:password  -nokeys \
-out /opt/cloudera/security/pki/hostname.pem
```

This extracted certificate can be used, as is, but you can additionally extract the private key from the keystore as detailed in the next section.

Extracting the Private Key from PKCS Keystore

Use OpenSSL to extract the private key from the PKCS keystore when needed. The command shown below extracts the key and saves it to a keystore that is protected using the password you provide:

```
   $ openssl pkcs12 -in /tmp/hostname-keystore.p12 -passin pass:password \
   -nocerts -out /opt/cloudera/security/pki/hostname.key -passout pass:password
```

To generate a key without a password, use this version of the command:

```
  $ openssl rsa -in /tmp/hostname-keystore.p12 -passin pass:password \
   -nocerts -out /opt/cloudera/security/pki/hostname.pem
```

Converting PEM Key and Certificate to JKS

Replace `hostname` in the commands below with the FQDN of the host whose certificate is being imported.

1. Convert the `openssl` private key and certificate files into a PKCS12 file.

```
$ openssl pkcs12 -export -in /opt/cloudera/security/pki/hostname.pem \
-inkey /opt/cloudera/security/pki/hostname.key -out /tmp/hostname.p12 \
-name hostname -passin pass:password -passout pass:password
```

2. Import the PKCS12 file into the Java keystore.

```
$ keytool -importkeystore -srckeystore /tmp/hostname.p12 -srcstoretype PKCS12 \
-srcstorepass password -alias hostname -deststorepass password \
-destkeypass password -destkeystore /opt/cloudera/security/jks/hostname-keystore.jks
```

How To Configure Authentication for Amazon S3

There are several ways to integrate Amazon S3 storage with Cloudera clusters, depending on your use case and other factors, including whether the cluster has been deployed using Amazon EC2 instances and if those instances were deployed using an IAM role, such as might be the case for clusters that have a single-user or small-team with comparable privileges. Clusters deployed to support many different users with various privilege levels to the Amazon S3 need to use AWS Credentials and have privileges to target data setup in Sentry. See How to Configure AWS Credentials on page 471 for details.

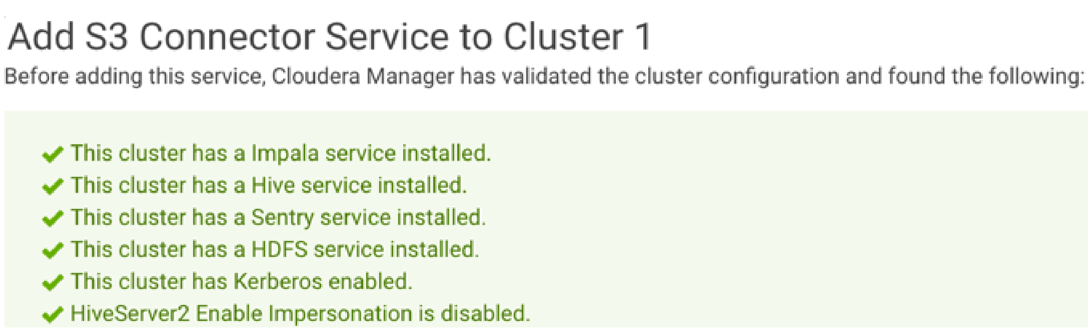
Authentication through the S3 Connector Service

Starting with CDH/Cloudera Manager 5.10, integration with Amazon S3 from Cloudera clusters has been simplified. Specifically, the **S3 Connector Service** automates the authentication process to Amazon S3 for Impala, Hive, and Hue, the components used for business-analytical use cases designed to run on persistent multi-tenant clusters.

The S3 Connector Service transparently and securely distributes AWS credentials needed by the cluster for the Amazon S3 storage. Access to the underlying Impala tables is controlled by Sentry role-based permissions. The S3 Connector Service runs on a secure cluster only, that is, a cluster configured to use:

- **Kerberos** for authentication, and
- **Sentry** for role-based authorization.

> **Note:** Of the items listed in the screenshot below, only the Sentry service and Kerberos being enabled are actual requirements. The other messages are for informational purposes only.



In Cloudera Manager 5.11, the S3 Connector Service set up wizard is launched automatically during the AWS Credential set up process when you select the path to add the S3 Connector Service.

See Configuring the Amazon S3 Connector for more information about the S3 Connector Service.

Authentication through Advanced Configuration Snippets

Before release 5.10 and the introduction of the S3 Connector Service, using Amazon S3 storage with the cluster involved adding the credentials to the `core-site.xml` configuration file (through Cloudera Manager's Advanced Configuration

Snippet mechanism). This approach is not recommended. AWS credentials provide read and write access to data stored on Amazon S3, so they should be kept secure at all times.

- Never share the credentials with other cluster users or services.
- Do not store in cleartext in any configuration files. When possible, use Hadoop's credential provider to encrypt and store credentials in the local JCEK (Java Cryptography Extension Keystore).
- Enable Cloudera sensitive data redaction to ensure that passwords and other sensitive information does not appear in log files.

> **Important:** Cloudera recommends using this approach for **single-user clusters** on **secure networks** only—networks that allow access only to authorized users, all of whom are also authorized to use the S3 credentials.

To enable CDH services to access Amazon S3, AWS credentials can be specified using the `fs.s3a.access.key` and `fs.s3a.secret.key` properties:

```
<property>
    <name>fs.s3a.access.key</name>
    <value>your_access_key</value>
</property>

<property>
    <name>fs.s3a.secret.key</name>
    <value>your_secret_key</value>
</property>
```

The process of adding AWS credentials is generally the same as that detailed in configuring server-side encryption for Amazon S3, that is, using the Cloudera Manager Admin Console to add the properties and values to the `core-site.xml` configuration file (Advanced Configuration Snippet). However, Cloudera strongly discourages this approach: in general, adding AWS credentials to the `core-site.xml` is not recommended. A somewhat more secure approach is to use temporary credentials, which include a session token that limits the viability of the credentials to a shorter time-frame within which a key can be stolen and used.

Using Temporary Credentials for Amazon S3

The AWS Security Token Service (STS) issues temporary credentials to access AWS services such as Amazon S3. These temporary credentials include an access key, a secret key, and a session token that expires within a configurable amount of time. Temporary credentials are not currently handled transparently by CDH, so administrators must obtain them directly from Amazon STS. For details, see Temporary Security Credentials in the AWS Identity and Access Management Guide.

> **Important:** Cloudera recommends using this approach only for **single-user clusters** on **secure networks**—networks that allow access only to authorized users, all of whom are also authorized to use the S3 credentials.

To connect to Amazon S3 using temporary credentials obtained from STS, submit them as command-line arguments with the Hadoop job. For example:

```
-Dfs.s3a.access.key=your_temp_access_key
-Dfs.s3a.secret.key=your_temp_secret_key
-Dfs.s3a.session.token=your_session_token_from_AmazonSTS
-Dfs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider
```

How to Configure Encryption for Amazon S3

Amazon offers several server-side encryption mechanisms for use with Amazon S3 storage. Cloudera clusters support server-side encryption for Amazon S3 data using either SSE-S3 (CDH 5.10 and later) or SSE-KMS (CDH 5.11 and later). With SSE-S3, keys are completely under the control of Amazon. With SSE-KMS, you have more control over the encryption keys, and can upload your own key material to use for encrypting Amazon S3. With either mechanism, encryption is applied transparently to the Amazon S3 bucket objects after you configure your cluster to use it.

Amazon S3 also supports TLS/SSL ('wire' or data-in-transit) encryption by default. Configuring data-at-rest encryption for Amazon S3 for use with your cluster involves some configuration both on Amazon S3 and for the cluster, using the Cloudera Manager Admin Console, as detailed below:

Requirements

Using Amazon S3 assumes that you have an Amazon Web Services account and the appropriate privileges on the AWS Management Console to set up and configure Amazon S3 buckets.

In addition, to configure Amazon S3 storage for use with a Cloudera cluster, you must have privileges as the User Administrator or Full Administrator on the Cloudera Manager Admin Console. See How to Configure AWS Credentials for details.

Amazon S3 and TLS/SSL Encryption

Amazon S3 uses TLS/SSL by default. Cloudera clusters (release 5.9 and later) include in their `core-site.xml` configuration file the boolean property, `fs.s3a.connection.ssl.enabled` set to `true` by default, which activates TLS/SSL. This means that if the cluster has been configured to use TLS/SSL, connections from the cluster to Amazon S3 automatically use TLS wire encryption for the communication.

> **Important:** Cloudera recommends that clusters always be configured for TLS/SSL. See How to Configure TLS Encryption for Cloudera Manager for details.

If the cluster is not configured to use TLS, the connection to Amazon S3 silently reverts to an unencrypted connection.

Amazon S3 and Data at Rest Encryption

Cloudera clusters can use either of these two Amazon S3 mechanisms for data-at-rest encryption:

- Server-side Encryption with AWS KMS-Managed Keys (SSE-KMS), which requires using Amazon Key Management Server (AWS KMS) in conjunction with your Amazon S3. You can have Amazon generate and manage the keys in AWS KMS for you, or you can provide your own key material, but you must configure AWS KMS and create a key before you can use it with your cluster. See Prerequisites for Using SSE-KMS on page 469 for details.
- Server-side Encryption with S3-Managed Encryption Keys (SSE-S3) , which is simplest to set up because it uses Amazon-provided and -managed keys and has no requirements beyond setting a single property. See Configuring the Cluster to Use Server-Side Encryption on Amazon S3 on page 469 for details.

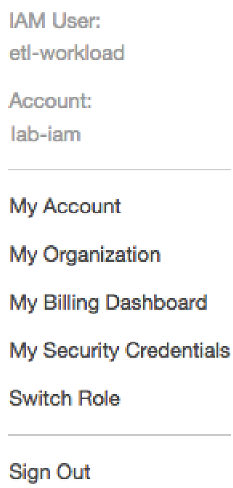
Enabling the cluster to use Amazon S3 server-side encryption involves using the Cloudera Manager Admin Console to configure the Advanced Configuration Snippet (Safety Valve) as detailed in Configuring the Cluster to Use Server-Side Encryption on Amazon S3 on page 469, below.

The steps assume that your cluster has been set up and that you have set up AWS credentials.

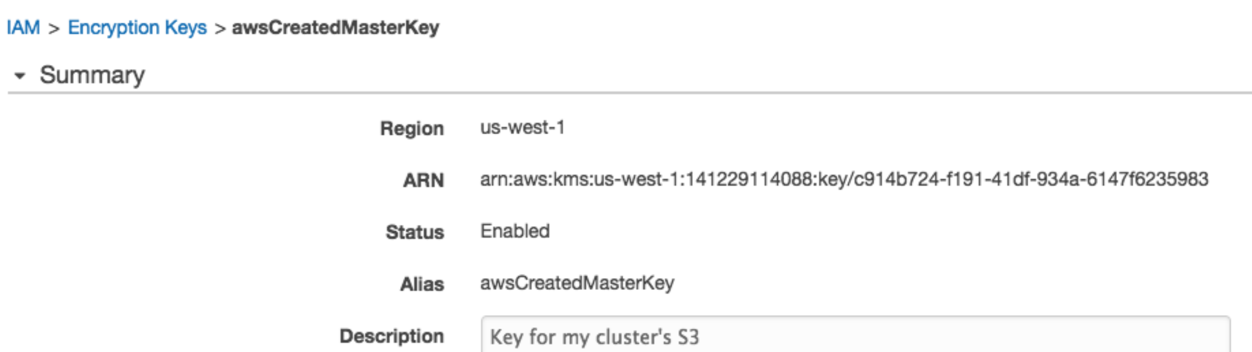
> **Note:** Cloudera clusters can be configured one encryption mode and key at a time for all objects in a given Amazon S3 bucket. However, you can change encryption modes or keys. See Changing Encryption Modes or Keys on page 470 for details.

Prerequisites for Using SSE-KMS

To use SSE-KMS with your Amazon S3 bucket, you must log in to the AWS Management Console using the account you set up in step 1 of Getting Started with Amazon Web Services on page 448. For example, the account *lab-iam* has an IAM user named *etl-workload* that has been granted permissions on the Amazon S3 storage bucket to be configured using SSE-KMS.



- Select **My Security Credentials** from the menu.
- Click **Encryption keys** (bottom left-hand on the AWS Management Console that displays at step 1, above).
- Click the **Create key** button to start the 5-step key-creation wizard that leads you through entry pages for giving the key an alias and description; adding tags, defining administrator permissions to the key, and defining usage permissions. The last page of the wizard shows you the policy that will be applied to the key before creating the key.



Configuring the Cluster to Use Server-Side Encryption on Amazon S3

Follow the steps below to enable server-side encryption on Amazon S3. To use SSE-KMS encryption, you will need your KMS key ID at step 7. Using SSE-S3 has no pre-requisites—Amazon generates and manages the keys transparently.

To configure the cluster to encrypt data stored on Amazon S3:

1. Log into the Cloudera Manager Admin Console.
2. Select **Clusters** > **HDFS**.
3. Click the **Configuration** tab.
4. Select **Scope** > **HDFS (Service-Wide)**.
5. Select **Category** > **Advanced**.
6. Locate the **Cluster-wide Advanced Configuration Snippet (Safety Valve) for core-site.xml** property.

7. In the text field, define the properties and values appropriate for the type of encryption you want to use.
   a. **To use SSE-S3 encryption:**

```
<property>
    <name>fs.s3a.server-side-encryption-algorithm</name>
    <value>AES256</value>
</property>
```

   b. **To use SSE-KMS encryption:**

```
<property>
    <name>fs.s3a.server-side-encryption-algorithm</name>
    <value>SSE-KMS</value>
</property>
<property>
    <name>fs.s3a.server-side-encryption-key</name>
    <value>your_kms_key_id</value>
</property>
```

8. Click **Save Changes**.
9. Restart the HDFS service.

For the value of *your_kms_key_id* (step 7b., above), you can use any of Amazon's four different key ID formats. Here are some examples:

Format	Example
Key ARN	arn:aws:kms:us-west-1:141229114088:key/c914b724-f191- 41df-934a-6147f6235983
Alias ARN	arn:aws:kms:us-west-1: 141229114088:key/c914b724-f191-41df-934a-6147f6235983: alias/awsCreatedMasterKey
Globally Unique Key ID	141229114088:key/c914b724-f191-41df-934a- 6147f6235983
Alias Name	alias/awsCreatedMasterKey

Changing Encryption Modes or Keys

Cloudera clusters can be configured to use only one type of server-side encryption for Amazon S3 data at a time.

However, Amazon S3 *does* support using different encryption keys for objects on an Amazon S3 bucket, which means that Amazon S3 continues to use whichever key was initially used to encrypt data (to decrypt on reads and re-encrypt on writes), even after a new mechanism and key exists. Your new key is used on new data written to the Amazon S3 bucket, while your 'old' key is used on any existing data that was encrypted with it. To summarize the behavior:

- Changing encryption mechanisms or keys on Amazon S3 has no effect on existing encrypted or unencrypted data.
- Data stored on Amazon S3 without encryption remains unencrypted even after you configure encryption for Amazon S3.
- Any existing encrypted data continues using the original mechanism and key to decrypt data (on reads) and re-encrypt data (on writes).
- After changing encryption mode or key, new objects stored on Amazon S3 from the cluster use the new mode and key.

Effect of Changing Encryption Mode or Key

This table shows the effect on existing encrypted or unencrypted data on Amazon S3 (far left column labeled "Data starts as...," reading down) and the result of "New" and "Existing" data and the keys that would be used after changing encryption-key configuration on the cluster. After changing encryption mode or key, existing data (Existing) and new data (New) use the mode and keys shown in columns 2 ("Unencrypted") through 5 ("Non-SSE Key"):

Data starts as...↓	Data results after modifying encryption mode or keys...			
	Unencrypted	**SSE-S3**	**SSE-KMS**	**Non-SSE Key**
Unencrypted	Existing	New	New	~
SSE-S3 encrypted	~	Existing	New	~
SSE-KMS [key1]	~	New	Existing [key1] New [key2]	~
Non-SSE key	~	~	~	Existing

Migrating Encrypted Data to New Encryption Mode or Key

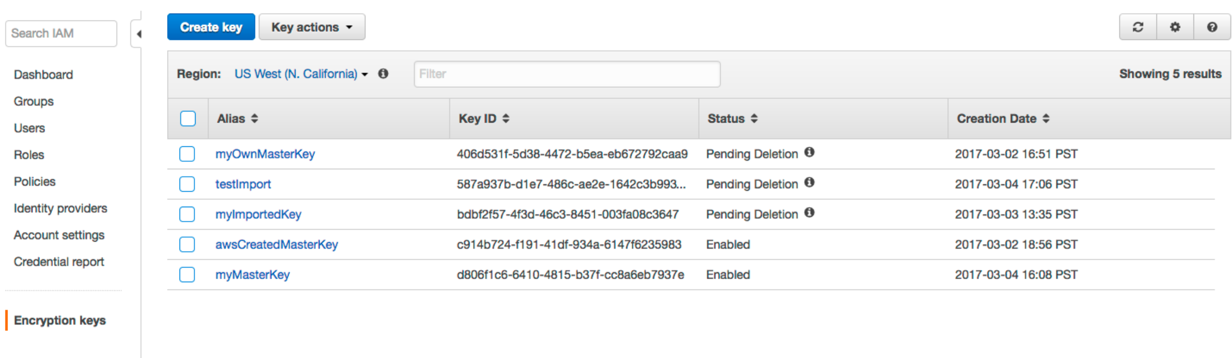
To use a new key with existing data (data stored on Amazon S3 using a different mechanism or key) you must first decrypt the data, then re-encrypt it using the new key. The process is as follows:

1. Create an Amazon S3 bucket as temporary storage for the unencrypted files.
2. Decrypt the data on the Amazon S3 bucket using the mechanism and key used for encryption (legacy encryption mode or key), moving the unencrypted data to the temporary bucket created in step 1.
3. Configure the Amazon S3 bucket to use the new encryption mechanism and key of your choice (SSE-S3, SSE-KMS)
4. Move the unencrypted data from the temporary bucket back to the Amazon S3 bucket that is now configured using the new mechanism and key.

Deleting an Encryption Key

If you change encryption modes or keys on Amazon S3, do not delete the key. To replace the old key and mode with a completely new mode or key, you should manually migrate the data.

When you delete an encryption key, Amazon puts the key in a **Pending Deletion** state (as shown in the Status column of the screenshot below) for at least 7 days, allowing you to reinstate a key if you change your mind or realize an error.



The pending time frame is configurable, from 7 up to 30 days. See AWS Key Management Service Documentation for complete details.

How to Configure AWS Credentials

Minimum Required Role: User Administrator (also provided by **Full Administrator**)

Amazon S3 (Simple Storage Service) can be used in a CDH cluster managed by Cloudera Manager in the following ways:

- As storage for Impala tables
- As a source or destination for HDFS and Hive/Impala replication and for cluster storage
- To enable Cloudera Navigator to extract metadata from Amazon S3 storage
- To browse S3 data using Hue

You can use the **S3Guard** feature to address possible issues with the "eventual consistency" guarantee provided by Amazon for data stored in S3. To use the S3Guard feature, you provision an Amazon DynamoDB that CDH uses as an additional metadata store to improve performance and guarantee that your queries return the most current data. See Configuring and Managing S3Guard.

To provide access to Amazon S3, you configure **AWS Credentials** that specify the authentication type (role-based, for example) and the access and secret keys. Amazon offers two types of authentication you can use with Amazon S3:

IAM Role-based Authentication

Amazon Identity and Access Management (IAM) can be used to create users, groups, and roles for use with Amazon Web Services, such as EC2 and Amazon S3. IAM role-based access provides the same level of access to all clients that use the role. All jobs on the cluster will have the same level of access to Amazon S3, so this is better suited for single-user clusters, or where all users of a cluster should have the same privileges to data in Amazon S3.

If you are setting up a peer to copy data to and from Amazon S3, using Cloudera Manager Hive or HDFS replication, select this option.

If you are configuring Amazon S3 access for a cluster deployed to Amazon Elastic Compute Cloud (EC2) instances using the IAM role for the EC2 instance profile, you do not need configure IAM role-based authentication for services such as Impala, Hive, or Spark.

> **Note:** IAM role-based authentication does not provide access to Amazon S3 using Cloudera Navigator.

Access Key Credentials

This type of authentication requires an AWS Access Key and an AWS Secret key that you obtain from Amazon and is better suited for environments where you have multiple users or multi-tenancy. You must enable the Sentry service and Kerberos when using the **S3 Connector** service. Enabling these services allows you to configure selective access for different data paths. (The Sentry service is not required for BDR replication or access by Cloudera Navigator.)

Cloudera Manager stores these values securely and does not store them in world-readable locations. The credentials are masked in the Cloudera Manager Admin console, encrypted in the configurations passed to processes managed by Cloudera Manager, and redacted from the logs.

For more information about Amazon S3, see the Amazon S3 documentation.

The client configuration files generated by Cloudera Manager based on configured services do not include AWS credentials. These clients must manage access to these credentials outside of Cloudera Manager. Cloudera Manager uses credentials stored in Cloudera Manager for trusted clients such as the Impala daemon and Hue. For access from YARN, MapReduce or Spark, see Using S3 Credentials with YARN, MapReduce, or Spark.

> **Note:** Isilon storage cannot be used with AWS credentials.

Adding AWS Credentials

Minimum Required Role: User Administrator (also provided by **Full Administrator**)

To add **AWS Credentials** for Amazon S3:

1. Open the Cloudera Manager Admin Console.
2. Click **Administration** > **External Accounts**.
3. Select the **AWS Credentials** tab.
4. Select one of the following:
    - **Add Access Key Credentials**

      This authentication mechanism requires you to obtain AWS credentials from Amazon.

      1. Enter a **Name** of your choosing for this account.

2. Enter the **AWS Access Key ID**.
3. Enter the **AWS Secret Key**.

- **Add IAM Role-Based Authentication**

1. Enter a name for your IAM Role-based authentication.

> **Note:** You cannot use IAM Role-based authentication for Cloudera Navigator access.

5. Click **Add**.

The **Edit S3Guard** dialog box displays.

S3Guard enables a consistent view of data stored in Amazon S3 and requires that you provision a DynamoDB database from Amazon Web Services. S3Guard is optional but can help improve performance and accuracy for certain types of workflows. To configure S3Guard, see Configuring and Managing S3Guard and return to these steps after completing the configuration.

If you do not want to enable S3Guard, click **Save** to finish adding the AWS Credential.

The **Connect to Amazon Web Services** dialog box displays.

6. Choose one of the following options:

- **Cloud Backup and Restore**

  To configure Amazon S3 as the source or destination of a replication schedule (to back up and restore data, for example), click the **Replication Schedules** link. See Data Replication for details.

- **Cluster Access to S3**

  To enable cluster access to S3 using the S3 Connector Service, click the **Enable for *Cluster Name*** link, which launches a wizard for adding the S3 Connector service. See Adding the S3 Connector Service for details.

- **Cloudera Navigator Access to S3**

  To give Cloudera Navigator access to Amazon S3, click the **Enable for Cloudera Navigator** link. Restart the Cloudera Navigator Metadata Server to enable access.

> **Note:** You cannot enable **Cloudera Navigator Access to S3** when using **IAM Role-based Authentication**.

Managing AWS Credentials

To remove AWS credentials:

> **Note:** You cannot remove AWS credentials if they are in use by a service in the cluster. You might need to edit the connectivity of the credential before removing it.

1. Open the Cloudera Manager Admin Console.
2. Click **Administration** > **External Accounts**.
3. Select the **AWS Credentials** tab.
4. Locate the row with the credentials you want to delete and click **Actions** > **Remove**.

To edit AWS Access Key credentials:

1. Open the Cloudera Manager Admin Console.

2. Click **Administration** > **External Accounts**.
3. Select the **AWS Credentials** tab.
4. Locate the row with the Access Key Credentials you want to delete and click **Actions** > **Edit Credential**.

   The **Edit Credential** dialog box displays.

5. Edit the account fields.
6. Click **Save**.
7. Restart cluster services that use these credentials. If connectivity is for Cloudera Navigator, restart the Cloudera Navigator Metadata server.

To rename the IAM Role-Based Authentication:

1. Open the Cloudera Manager Admin Console.
2. Click **Administration** > **External Accounts**.
3. Select the **AWS Credentials** tab.
4. Locate the row with the IAM Role-Based Authentication you want to rename and click **Actions** > **Rename**.
5. Enter a new name.
6. Click **Save**.

   The **Connect to Amazon Web Services** screen displays.

7. Click the links to change any service connections or click **Close** to leave them unchanged.

To edit the services connected to an AWS Credentials account:

1. Open the Cloudera Manager Admin Console.
2. Click **Administration** > **External Accounts**.
3. Select the **AWS Credentials** tab.
4. Locate the row with the credentials you want to edit and click **Actions** > **Edit Connectivity**.

   The **Connect to Amazon Web Services** screen displays.

5. Click one of the following options:

   - **Cloud Backup and Restore**

     To configure Amazon S3 as the source or destination of a replication schedule (to back up and restore data, for example), click the **Replication Schedules** link. See Data Replication for details.

   - **Cluster Access to S3**

     To enable cluster access to S3 using the S3 Connector Service, click the **Enable for *Cluster Name*** link, which launches a wizard for adding the S3 Connector service. See Adding the S3 Connector Service for details.

   - **Cloudera Navigator Access to S3**

     To give Cloudera Navigator access to Amazon S3, click the **Enable for Cloudera Navigator** link. Restart the Cloudera Navigator Metadata Server to enable access.

     > **Note:** You cannot enable **Cloudera Navigator Access to S3** when using **IAM Role-based Authentication**.

How to Enable Sensitive Data Redaction

Redaction is a process that obscures data. It helps organizations comply with government and industry regulations, such as PCI (Payment Card Industry) and HIPAA, by making personally identifiable information (PII) unreadable except to those whose jobs require such access. For example, in simple terms, HIPAA legislation requires that patient PII is available only to appropriate medical professionals (and the patient), and that any medical or personal information exposed outside the appropriate context cannot be used to associate an individual's identity with any medical information. Data redaction can help ensure this privacy, by transforming PII to meaningless patterns—for example, transforming U.S. social security numbers to XXX-XX-XXXX strings.

Data redaction works separately from Cloudera data encryption techniques. Data encryption alone does not preclude administrators with full access to the cluster from viewing sensitive user data. Redaction ensures that cluster administrators, data analysts, and others cannot see PII or other sensitive data that is not within their job domain. At the same time, it does not prevent users with appropriate permissions from accessing data to which they have privileges.

Cloudera clusters implement some redaction features by default, while some features are configurable and require administrators to specifically enable them. The details are covered below:

Cloudera Manager and Passwords

As of Cloudera Manager 5.5 (and later releases) passwords are no longer stored in cleartext, neither through the Cloudera Manager Admin Console nor in the configuration files on disk. Passwords managed by Cloudera Manager and Cloudera Navigator are redacted internally, with the following results:

- In the Cloudera Manager Admin Console:
  - In the **Processes** page for a given role instance, passwords in the linked configuration files are replaced by `*******`.
  - **Advanced Configuration Snippet (Safety Valve)** parameters, such as passwords and secret keys, are visible to users (such as admins) who have edit permissions on the parameter, while those with read-only access see redacted data. However, the parameter name is visible to anyone. (Data to be redacted from these snippets is identified by a fixed list of key words: *password*, *key*, *aws*, and *secret*.)
- On all Cloudera Manager Server and Cloudera Manager Agent hosts:
  - Passwords in the configuration files in `/var/run/cloudera-scm-agent/process` are replaced by `********`.

Cloudera Manager Server Database Password Handling

Unlike the other passwords that are redacted or encrypted by Cloudera Manager, the password used for the Cloudera Manager Server database is stored in plaintext in the configuration file, `/etc/cloudera-scm-server/db.properties`, as shown in this example:

```
# Auto-generated by scm_prepare_database.sh on Mon Jan 30 05:02:18 PST 2017
#
# For information describing how to configure the Cloudera Manager Server
# to connect to databases, see the "Cloudera Manager Installation Guide."
#
com.cloudera.cmf.db.type=mysql
com.cloudera.cmf.db.host=localhost
com.cloudera.cmf.db.name=cm
com.cloudera.cmf.db.user=cm
com.cloudera.cmf.db.setupType=EXTERNAL
com.cloudera.cmf.db.password=password
```

However, as of Cloudera Manager 5.10 (and higher), rather than using a cleartext password you can use a script or other executable that uses `stdout` to return a password for use by the system.

During installation of the database, you can pass the script name to the `scm_prepare_database.sh` script with the `--scm-password-script` parameter. See Setting up the Cloudera Manager Server Database and Syntax for scm_prepare_database.sh for details.

You can also replace an existing cleartext password in `/etc/cloudera-scm-server/db.properties` by replacing the `com.cloudera.cmf.db.password` setting with `com.cloudera.cmf.db.password_script` and setting the name of the script or executable:

Cleartext Password (5.9 and prior)	Script (5.10 and higher)
com.cloudera.cmf.db.password=*password*	com.cloudera.cmf.db.password_script=*script_name_here*

At runtime, if `/etc/cloudera-scm-server/db.properties` does not include the script identified by `com.cloudera.cmf.db.password_script`, the system looks for the value of `com.cloudera.cmf.db.password`.

Log and Query Redaction

Cloudera Manager provides a configurable log and query redaction feature that lets you redact sensitive data in the CDH cluster as it's being written to the log files (see the Cloudera Engineering Blog "Sensitive Data Redaction" post for a technical overview), to prevent leakage of sensitive data. Redaction works only on data, not metadata—that is, sensitive data inside files is redacted, but the name, owner, and other metadata about the file is not.

Redaction is enabled for the entire cluster through the Cloudera Manager Admin Console, which also lets you define rules to target sensitive data in SQL data and log files. After enabling data redaction, the following contain replacement strings (such as a series of Xs) for the sensitive data items you define in your rules:

- Logs in HDFS and any dependent cluster services.
- Audit data sent to Cloudera Navigator.
- SQL query strings displayed by Hue, Hive, and Impala.

> **Note:** Log redaction is not available in Isilon-based clusters.

See Enabling Log and Query Redaction Using Cloudera Manager on page 477 (below) for information about how to enable and define rules for sensitive data redaction for your cluster's logs and SQL queries (Hive, Hue, Impala).

How Redaction Rules Work

Cloudera's redaction process (redactor) uses regular expressions to target data for redaction. Common regular expression patterns for sensitive data include social security numbers, credit card numbers, email addresses, and dates, for example. The redaction rules are specified using the following elements:

- **Search -** Regular expression to compare against the data. For example, the regular expression `\d{4}[^\w]\d{4}[^\w]\d{4}[^\w]\d{4}` searches for a credit card number pattern. Segments of data that match the regular expression are redacted using the Replace string.
- **Replace -** String used to redact (obfuscate) data, such as a pattern of Xs to replace digits of a credit card number: `XXXX-XXXX-XXXX-XXXX`.
- **Trigger -** Optional simple string to be searched before applying the regular expression. If the string is found, the redactor searches for matches using the Search regular expression. Using the Trigger field improves performance: simple string matching is faster than regular expression matching.

You can use the following preconfigured redaction rules on your cluster. Rules are applied in the order listed in the table.

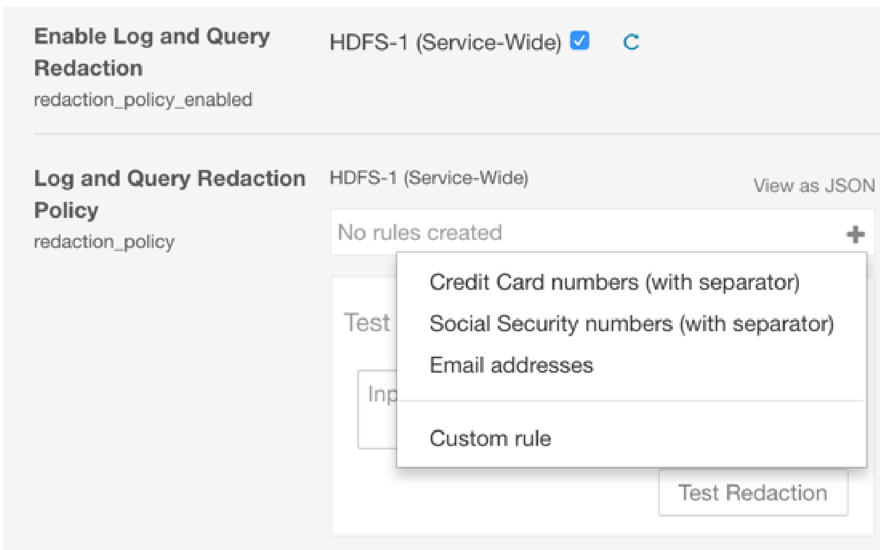
Rule	Regex Pattern	Replacement
Credit Card numbers (with separator)	`\d{4}[^\w]\d{4}[^\w]\d{4}[^\w]\d{4}`	`XXXX-XXXX-XXXX-XXXX`
Social Security numbers (with separator)	`\d{3}[^\w]\d{2}[^\w]\d{4}`	`XXX-XX-XXXX`
Email addresses	`\b([A-Za-z0-9]|[A-Za-z0-9][A-Za-z0-9\-\._]\ *[A-Za-z0-9])@(([A-Za-z0-9]|[A-Za-z] \ [A-Za-z0-9\-]*[A-Za-z0-9])\.)+([A-Za-z0-9]\ |[A-Za-z0-9][A-Za-z0-9\-]*[A-Za-z0-9])\b`	`email@redacted.host`

<table>
<tr><th>Rule</th><th>Regex Pattern</th><th>Replacement</th></tr>
<tr><td>Hostnames</td><td>\b(([A-Za-z]|[A-Za-z][A-Za-z0-9\-] \
*[A-Za-z0-9])\.)+([A-Za-z0-9] \
|[A-Za-z0-9][A-Za-z0-9\-]*[A-Za-z0-9])\b</td><td>HOSTNAME.REDACTED</td></tr>
</table>

Enabling Log and Query Redaction Using Cloudera Manager

To enable redaction, you must use Cloudera Manager's new layout rather than the classic layout (the Switch to the new layout, Switch to the classic layout links toggle between these two variations of the UI). To enable log and query redaction in Cloudera Manager:

1. Login to the Cloudera Manager Admin Console.
2. Select **Clusters** > **HDFS**.
3. Click the **Configuration** tab.
4. In the **Search** box, type `redaction` to find the redaction property settings:
    - Enable Log and Query Redaction
    - Log and Query Redaction Policy List of rules for redacting sensitive information from log files and query strings. Choose a preconfigured rule or add a custom rule. See How Redaction Rules Work on page 476 for more information about rule pattern definitions.

    Test your rules:

    - Enter sample text into the Test Redaction Rules text box
    - Click Test Redaction.

5. Click **Save Changes**.
6. Restart the cluster.

If no rules match, the text you entered displays in the Results field, unchanged.

Using Cloudera Navigator Data Management for Data Redaction

Important: This approach has been supplanted by Cloudera Manager's cluster-wide log and query redaction feature, and is not recommended.

You can specify credit card number patterns and other PII to be masked in audit events, in the properties of entities displayed in lineage diagrams, and in information retrieved from the Audit Server database and the Metadata Server persistent storage. Redacting data other than credit card numbers is not supported by default with the Cloudera

Navigator property. You can use regular expressions to redact social security numbers or other PII. Masking applies only to audit events and lineage entities generated after enabling a mask.

Required Role: Navigator Administrator or Full Administrator

1. Log into **Cloudera Manager Admin Console**.
2. Select **Clusters** > **Cloudera Management Service**.
3. Click the **Configuration** tab.
4. Expand the **Navigator Audit Server Default Group** category.
5. Click the **Advanced** category.
6. Configure the **PII Masking Regular Expression** property with a regular expression that matches the credit card number formats to be masked. The default expression is:

```
(4[0-9]{12}(?:[0-9]{3})?)|(5[1-5][0-9]{14})|
(3[47][0-9]{13})|(3(?:0[0-5]|[68][0-9])[0-9]{11})|
(6(?:011|5[0-9]{2})[0-9]{12})|((?:2131|1800|35\\d{3})\\d{11})
```

   which consolidates these regular expressions:

   - Visa - `(4[0-9]{12}(?:[0-9]{3})?)`
   - MasterCard - `(5[1-5][0-9]{14})`
   - American Express - `(3[47][0-9]{13})`
   - Diners Club - `(3(?:0[0-5]|[68][0-9])[0-9]{11})`
   - Discover - `(6(?:011|5[0-9]{2})[0-9]{12})`
   - JCB - `((?:2131|1800|35\\d{3})\\d{11})`

   If the property is left blank, PII information is not masked.
7. Click **Save Changes**.

How To Enable Sentry High Availability

This article walks you through the steps to enable high availability for Sentry in Cloudera Manager using a rolling restart. With high availability enabled, if one of the Sentry hosts becomes unavailable, the client can connect to the other host. To enable high availability, you will perform the following tasks:

1. Verify the prerequisites.
2. Choose an additional Sentry host.
3. Perform a rolling restart on the cluster.

Using a rolling restart instead of manually shutting down the cluster minimises downtime. Note that even with a rolling restart, the Sentry service stops and restarts. For more information about using Cloudera Manager to perform a rolling restart, see Rolling Restart.

For more information about Sentry high availability, see Sentry High Availability on page 375.

Verify the Prerequisites

Verify the following prerequisites before you enable high availability for Sentry:

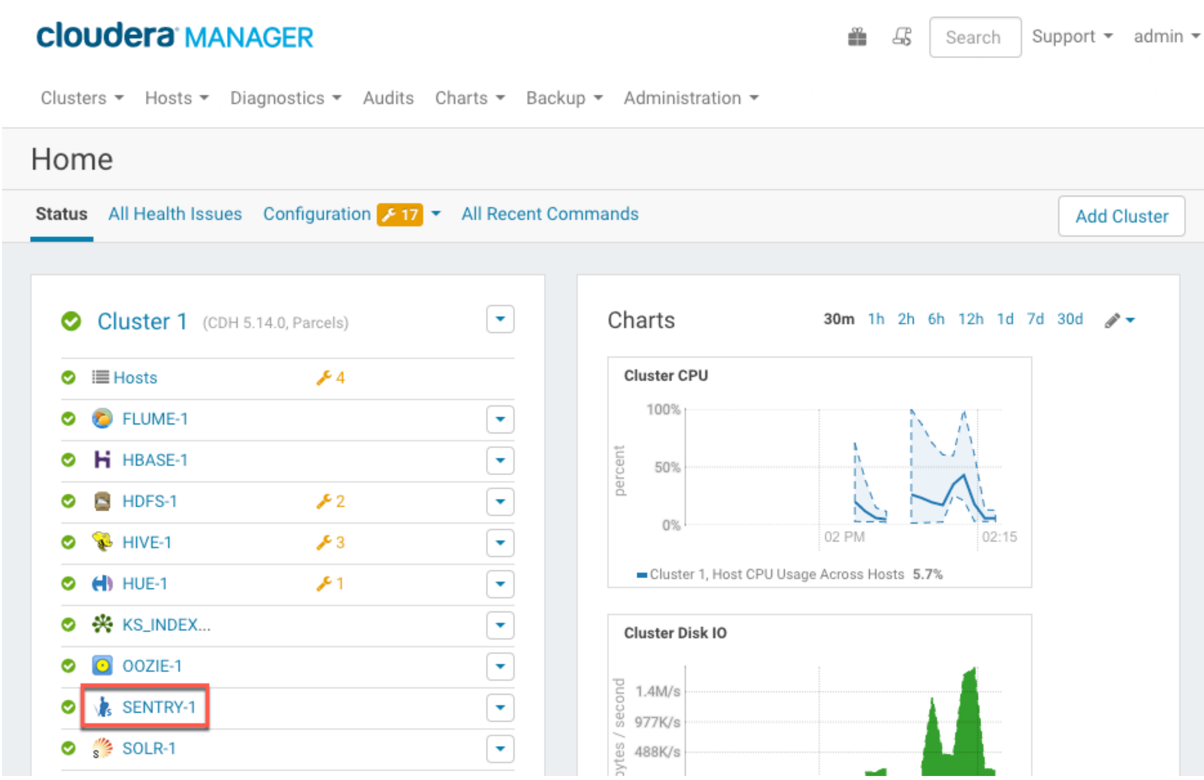
- You must have CDH version 5.13.0 or higher installed.
- You must have Cloudera Manager version 5.13.0 or higher installed.
- The Sentry service must be installed and running. See Before You Install Sentry on page 359 and Installing and Upgrading the Sentry Service on page 360 for information about how to install the Sentry service.
- The ZooKeeper service must be enabled. To view the ZooKeeper service that the Sentry service depends on, open the Sentry service in Cloudera Manager, click the **Configuration** tab and enter **ZooKeeper** in the **Search** box. The **ZooKeeper Service** property is displayed.
- The Sentry service must be backed by a relational database, not a flat file.

- You must have HDFS high availability enabled. For information about configuring HDFS high availability, see HDFS High Availability.

Enable High Availability

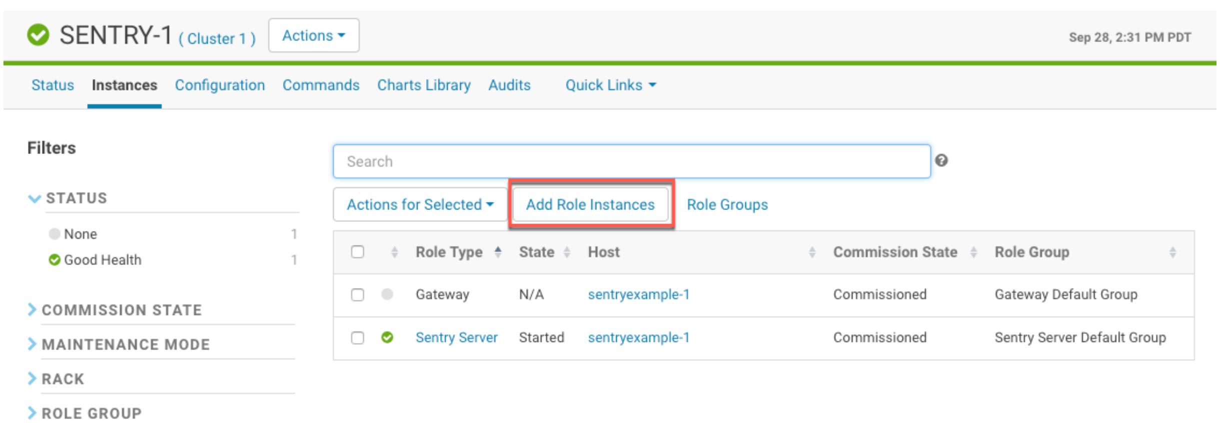
Complete the following steps to enable Sentry high availability with a rolling restart:

1. In Cloudera Manager, open the Sentry service. The following image shows the location of the Sentry service on an example cluster. Click the Sentry service name to open the service.



2. Open the **Instances** tab and click **Add Role Instances**.

   The following image shows the location of the **Add Role Instances** button:



3. In the **Add Role Instances** wizard, click the **Select a host** button for the Sentry Server. You do not have to edit the Gateway host.
4. A pop-up window opens that allows you to pick an alternate host for the Sentry service. Click the hostname of the host that you want to use for high availability. The host that the Sentry service is currently running on is already selected and grayed out. You can select one alternate host for the Sentry service.

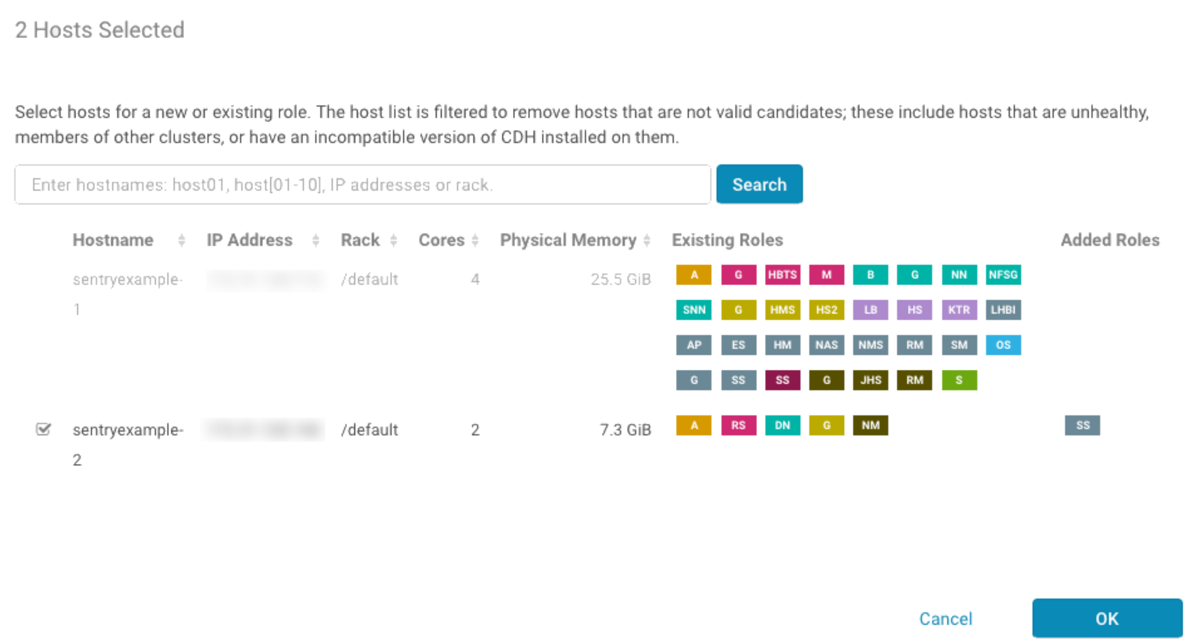
You can search for a host or you can filter the list by entering a range of hosts in the **Search** field. You can search for hosts with the following information:

- Range of hostnames. Use the following range definitions:

Range Definition	Matching Hosts
10.1.1.[1-4]	10.1.1.1, 10.1.1.2, 10.1.1.3, 10.1.1.4
host[1-3].company.com	host1.company.com, host2.company.com, host3.company.com
host[07-10].company.com	host07.company.com, host08.company.com, host09.company.com, host10.company.com

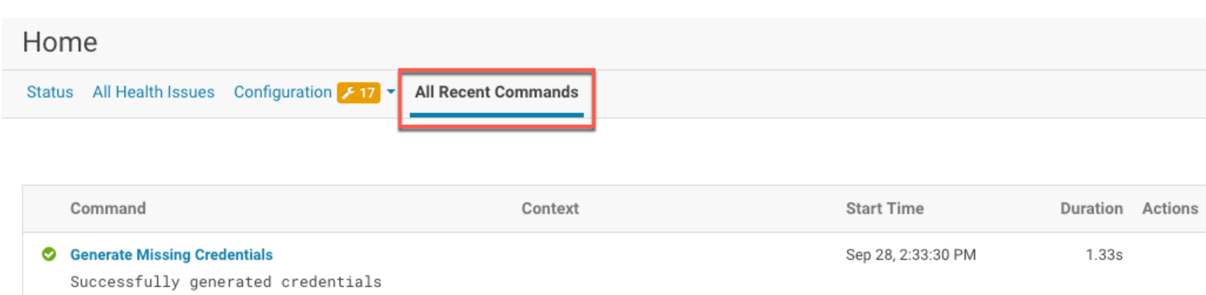
- IP addresses
- Rack name

The following image shows the host options. In this example, `sentryexample-1` is the current host and we are adding `sentryexample-2` as the alternate host.



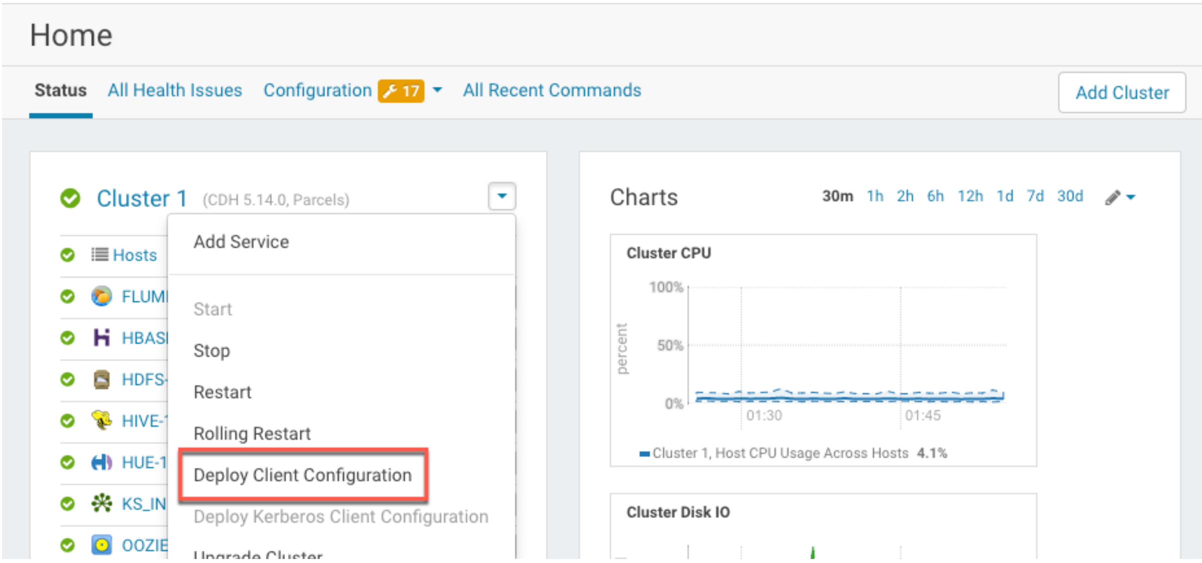
5. After you have selected the alternate host, click **OK** to close the window. The hostname appears below the Sentry Server.
6. Click **Continue**. The wizard closes and the Sentry hosts are listed.
7. Go to the Cloudera Manager home page and click the **All Recent Commands** tab. Verify that the **Generate Missing Credentials** command has successfully completed before you proceed with the rolling restart.

   The following image shows the **All Recent Commands** tab with and the **Generate Missing Credentials** command successfully completed:



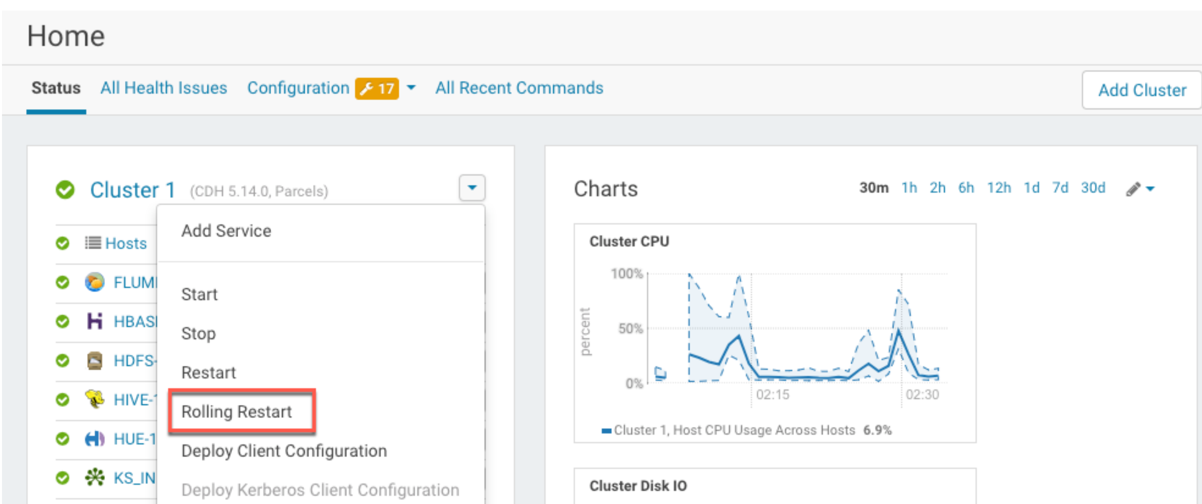
8. Click the **Status** tab to return to the Cloudera Manager home page.
9. Click the down arrow next to the cluster name to open the menu and click **Deploy Client Configuration**.

   The following image shows the location of **Deploy Client Configuration** in the menu:



10. A window opens asking if you are sure you want to run the Deploy Client Configuration command. Click **Deploy Client Configuration**.
11. When the steps are successfully completed, click **Close** to close the window.
12. Click the down arrow next to the cluster name again to open the menu and click **Rolling Restart**. For detailed information about the rolling restart, see Rolling Restart.

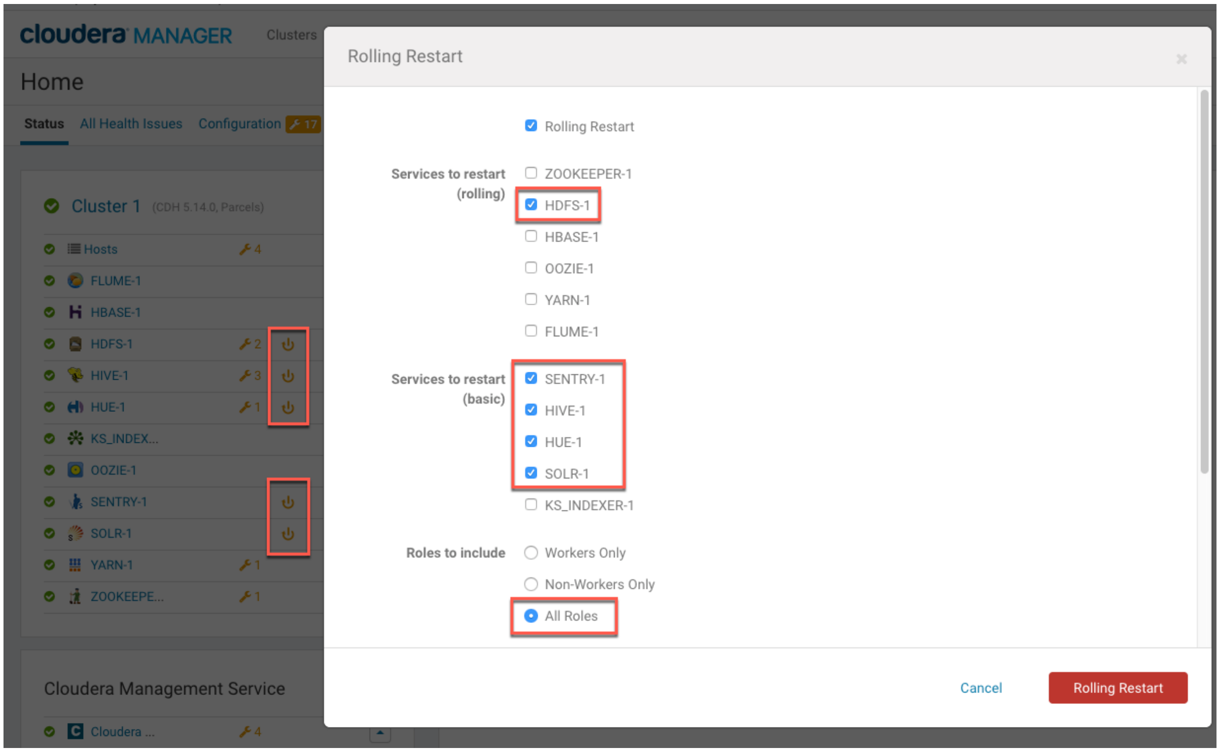
    The following image shows the location of **Rolling Restart** in the menu:



13. In the **Rolling Restart** window that opens, select the services that have stale configurations, including the Sentry service. Every service that is dependent on Sentry will have a stale configuration. In the **Roles to Include** parameter, select **All Roles**.

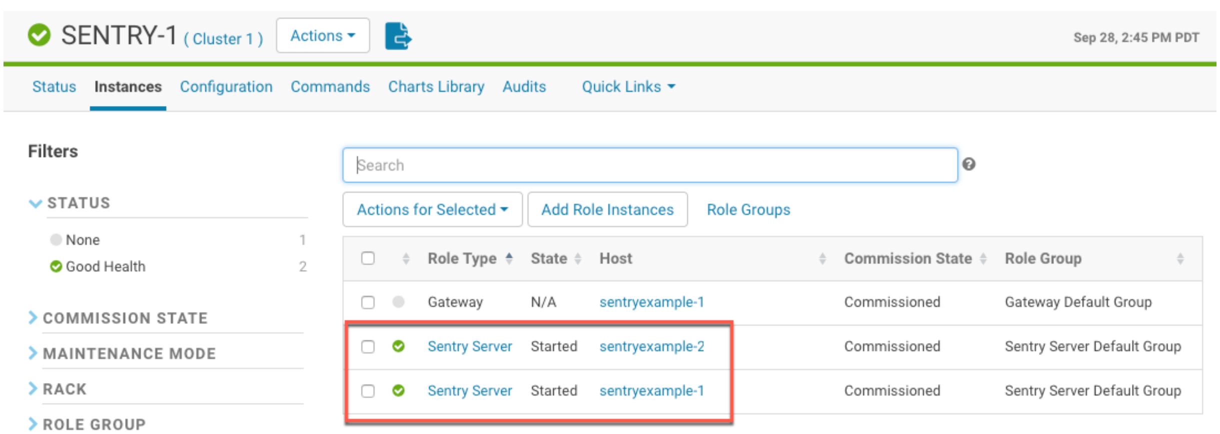
    Note that the services that do not support rolling restart will simply be restarted and will be unavailable during their restart, including the Sentry service.

    The following image shows the example cluster with the services that have stale configurations selected, as well as the **All Roles** option. The red boxes also show the location of the stale configuration markers in Cloudera Manager.



14. Click **Rolling Restart** to begin the restart.
15. A pop-up window opens that lists the rolling restart steps and the status of each step. When the rolling restart is complete, click **Close** to close the pop-up.
16. Verify that none of the services have stale configurations. If one or more services has a stale configuration after the rolling restart, Sentry high availability might not be configured properly.
17. To verify the statuses of the Sentry hosts, open the Sentry service and click the **Instances** tab.

    The following image shows the example cluster with Sentry high availability enabled. There are two Sentry hosts running.



How to Log a Security Support Case

Before logging a support case, collect as much information about your issue as possible, as detailed below:

- If possible, gather a diagnostic bundle.
- Note specific details about the issue, including these:

- Is this a new install, or did you upgrade a working cluster? For upgrades, identify release numbers ("trying to upgrade from Cloudera Manager 5.5.6 to Cloudera Manager 5.8.1," for example).
- Is this a production deployment, development environment, or sandbox environment?
- Note the severity of the impact and whether it is causing a production outage.
- Capture error messages (screenshots, command-line capture, and so on) and attach to the case.
- Note the commands executed and the results, captured to a file if possible.
- Itemize all changes made to your cluster configuration after the issue arose (possibly, as attempts to resolve the issue).

The following are some additional details to gather when contacting support:

- Kerberos Issues on page 483
- LDAP Issues on page 483
- TLS/SSL Issues on page 483

Kerberos Issues

- For Kerberos issues, your `krb5.conf` and `kdc.conf` files are valuable for support to be able to understand your configuration.
- If you are having trouble with client access to the cluster, provide the output for `klist -ef` after kiniting as the user account on the client host in question. Additionally, confirm that your ticket is renewable by running `kinit -R` after successfully kiniting.
- Specify if you are authenticating (kiniting) with a user outside of the Hadoop cluster's realm (such as Active Directory, or another MIT Kerberos realm).
- If using AES-256 encryption, ensure you have the Unlimited Strength JCE Policy Files deployed on all cluster and client nodes.

LDAP Issues

- Specify the LDAP service in use (Active Directory, OpenLDAP, one of Oracle Directory Server offerings, OpenDJ, etc)
- Provide a screenshot of the LDAP configuration screen you are working with if you are troubleshooting setup issues.
- Be prepared to troubleshoot using the `ldapsearch` command (requires the `openldap-clients` package) on the host where LDAP authentication or authorization issues are being seen.

TLS/SSL Issues

- Specify whether you are using a private/commercial CA for your certificates, or if they are self-signed. Note that Cloudera strongly recommends against using self-signed certificates in production clusters.
- Clarify what services you are attempting to setup TLS/SSL for in your description.
- When troubleshooting TLS/SSL trust issues, provide the output of the following `openssl` command:

```
openssl s_client -connect host.fqdn.name:port
```

How To Obtain and Deploy Keys and Certificates for TLS/SSL

Cloudera recommends obtaining certificates from one of the trusted public certificate authorities (CA) such as Symantec or Comodo for TLS/SSL encryption for the cluster. This How To provides a brief overview of the tools used to create the various artifacts followed by the steps in the process. Plan ahead to allow time to receive signed certificates from whichever CA you use.

> **Note:** Always check with your selected public CA before creating any CSRs and follow the CA-specific processes.

If your organization uses its own internal CA, follow your internal process for creating and submitting the CSRs.

Tools Overview

Java Keytool and OpenSSL are key management tools that let you create the security artifacts needed for TLS/SSL. See How to Convert PEM to JKS and JKS to PEM for TLS/SSL Services and Clients for more information beyond the two short overviews below.

Java Keytool

Oracle Java `keytool` is a utility included with the Oracle JDK for creating and managing cryptographic keys and certificates. During configuring the Cloudera Manager Cluster for TLS/SSL, you create the private key pairs, keystore, certificate signing requests, and create a truststore for specific use by the cluster using this software tool, as detailed in the steps throughout this guide.

Java Keytool Requirements for Cloudera Manager TLS/SSL Configuration

For any steps using the Java Keytool, be sure to:

- Use the Oracle Java `keytool` rather than tools such as OpenJDK.
- Use the JDK downloaded from Oracle or the Cloudera-provided Oracle JDK located in this default path on a Cloudera Manager Server host:

```
/usr/java/jdk1.7.0_67-cloudera/bin/jre/lib/security
```

- Use the same version of the Java keytool for all steps. If the host has multiple JDKs installed, set the PATH variable so that the Oracle JDK is invoked first, as in this example:

```
$ export JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera
$ export PATH=$JAVA_HOME/bin:$PATH
```

- Use the same *password* for the `-keypass` and `-storepass` in any commands that invoke these two options. Cloudera Manager requires the same password for a key and its keystore.

OpenSSL

OpenSSL is an open source cryptography and TLS/SSL toolkit that has been widely used since its inception ~ 1999. Just as with Java Keytool, OpenSSL lets you create private keys, certificate requests, and keystores, and it provides options for verifying certificates.

Cloudera Manager Agent hosts act as clients of a Cloudera Manager Server host during RPC client and server communications. The Agent hosts, Hue, Impala and other Python-based services require PEM-formatted keys and certificates (PKCS #8), which is why the steps below include converting some of the JKS artifacts using this tool. See How to Convert PEM to JKS and JKS to PEM for TLS/SSL Services and Clients for more information.

Step 1: Create Directory for Security Artifacts

Distributing the certificates, keys, truststore—in short, all security artifacts used for TLS/SSL intra-cluster communications—is part of this and some subsequent processes. To keep things organized, Cloudera recommends that you create the directory and distribute and store artifacts when you receive them, even though they may not be immediately needed.

The following table shows an example directory structure for the security artifacts created in the steps in this and subsequent sections. Use different names if you like, but for ease of deployment, use the same directory names across all hosts in the cluster.

Example	Description
cmhost.sec.example.com	FQDN of an example Cloudera Manager Server host.
/opt/cloudera/security	Base path for security-related files.
/opt/cloudera/security/pki	Path for all security artifacts associated with TLS/SSL, including keys, keystores (*keystore*.jks), CSR, and root- and intermediate-CA certificates.
/usr/java/jdk1.7.0_67-cloudera/jre/lib/security/jssecacerts	Path to the default alternative Java truststore on a Cloudera Manager Server host system.

1. On the Cloudera Manager Server host, create the `/opt/cloudera/security/pki` directory:

```
$ sudo mkdir -p /opt/cloudera/security/pki
```

2. This directory must be owned by `cloudera-scm:cloudera-scm` and have its executable bit set correctly (umask 700 or umask 740) so that Cloudera Manager can access the keystore at runtime:

```
$ sudo chown -R cloudera-scm:cloudera-scm /opt/cloudera/security/jks
$ sudo umask 700
$ cd /opt/cloudera/security/jks
```

Directories and artifacts persist during system upgrades. For security purposes, for any host you remove from a cluster, remove any directories you create and more importantly, remove any security artifacts (keys, certificates, and so on) they may contain.

Step 2: Create the Java Truststore

On the Cloudera Manager Server host, copy the JDK `cacerts` file to `jssecacerts`:

```
$ sudo cp $JAVA_HOME/jre/lib/security/cacerts $JAVA_HOME/jre/lib/security/jssecacerts
```

If you do not have the `$JAVA_HOME` variable set, replace it with the path to the Oracle JDK. For example, the default path to the Java JDK on a Cloudera Manager Server host is:

`/usr/java/jdk1.7.0_67-cloudera/`

Step 3: Generate Server Key and CSR

1. On the Cloudera Manager Server host, use the `keytool` utility to generate a keypair and a keystore named for the server. Replace the OU, O, L, ST, and C entries with the values for your organization name, location, and country code (US, in the example):

```
$ sudo keytool -genkeypair -alias $(hostname -f)-server -keyalg RSA -keystore \
/opt/cloudera/security/pki/$(hostname -f)-server.jks -keysize 2048 -dname \
"CN=$(hostname -f),OU=Dept,O=Example.com,L=City,ST=State,C=US" \
-storepass password -keypass password
```

   Use the same *password* for the key and its keystore (`-keypass` and `-storepass`, respectively): Cloudera Manager does not support using different passwords for the key and keystore.

2. Generate a certificate signing request (CSR) for the public key (contained in the keystore as a result of the command above). In this command below, enter the password that you set for the key and the keystore in the command above:

```
$ sudo keytool -certreq -alias $(hostname -f)-server \
-keystore /opt/cloudera/security/pki/$(hostname -f)-server.jks \
-file /opt/cloudera/security/pki/$(hostname -f)-server.csr -storepass password \
-keypass password
```

Step 4: Submit the CSR to the CA

1. Submit the CSR file to your certificate authority using the process and means required by the CA, for example, email or web submission. For the certificate format, specify PEM (base64 ASCII) (see Step 5 below for an example of PEM formatted certificate heading and closing structure).
2. The public CA will request specific details from you, to verify that you own the domain name contained in the CSR, before they issue the certificate.
3. When you receive the signed certificate from the CA, you can proceed with Step 5.

Step 5: Verify the Certificate

From the public (or internal) CA, you receive the certificate for the server signed by the CA and several other digital artifacts, including root CA and possibly one (or more) intermediate CA certificates. Before distributing these to the hosts, make sure the certificate is in PEM format. A PEM formatted certificate file looks like this:

```
-----BEGIN CERTIFICATE-----
<The encoded certificate is represented by multiple lines of exactly 64 characters,
except
for the last line, which can contain 64 characters or fewer.>
-----END CERTIFICATE-----
```

If your issued certificate is in binary (DER) format, convert it to PEM format before continuing. See How to Convert Formats (PEM, JKS) for TLS/SSL Clients and Services for details.

To modify the truststore (jssecacerts) to explicitly trust certificates or certificate chain (as you might for certificates signed by an internal CA), follow the steps in How to Add Root and Intermediate CAs to Truststore for TLS/SSL.

Step 6: Import the Certificate into the Keystore

Copy the signed certificate you receive from the CA to the Cloudera Manager Server host. To identity the certificate's functionality, include a suffix such as "-server.cert.pem" in the target name for the copy command, as shown in this example:

```
cp cert-file-recd /opt/cloudera/security/pki/$(hostname -f)-server.cert.pem
```

Import the certificate into the keystore.

```
$ sudo keytool -importcert -alias $(hostname -f)-server \
-file /opt/cloudera/security/pki/$(hostname -f)-server.cert.pem \
-keystore /opt/cloudera/security/pki/$(hostname -f)-server.jks
```

Assuming the certificate was obtained from a public CA, you can safely disregard this message about trust, and enter yes to continue:

```
... is not trusted. Install reply anyway? [no]:  yes
```

You *must* see the following response confirming that the certificate has been properly imported and can verify the private key that it certifies.

```
Certificate reply was installed in keystore
```

If you do not see this response, double-check all your steps up to this point: are you working in the correct path? Do you have the proper certificate? and so on. See Getting Support for information about how to contact Cloudera Support and to find out about other sources of help if you cannot successfully import the certificates.

How to Set Up a Gateway Node to Restrict Access to the Cluster

The steps below configure a firewall-protected Hadoop cluster that allows access only through the node configured as the gateway. Clients access the cluster through the gateway using the REST API, for example, using HttpFS (which provides REST access to HDFS) or using Oozie, which allows REST access for submitting and monitoring jobs.

Installing and Configuring the Firewall and Gateway

Follow these steps:

1. Choose a cluster node to be the gateway machine
2. Install and configure the Oozie server by following the standard directions starting here: Installing Oozie
3. Install HttpFS.
4. Start the Oozie server:

```
$ sudo service oozie start
```

5. Start the HttpFS server:

```
$ sudo service hadoop-httpfs start
```

6. Configure firewalls.

   Block all access from outside the cluster.

   - The gateway node should have ports 11000 (oozie) and 14000 (hadoop-httpfs) open.
   - Optionally, to maintain access to the Web UIs for the cluster's JobTrackers and NameNode, open their HTTP ports: see Ports Used by Components of CDH 5.
7. Optionally configure authentication in simple mode (default) or using Kerberos. See HttpFS Authentication on page 134 to configure Kerberos for HttpFS and Oozie Authentication on page 149 to configure Kerberos for Oozie.
8. Optionally encrypt communication using HTTPS for Oozie by following these directions.

Accessing HDFS

With the Hadoop client:

All of the standard `hadoop fs` commands work; just make sure to specify `-fs webhdfs://HOSTNAME:14000`. For example (where *GATEWAYHOST* is the hostname of the gateway machine):

```
$ hadoop fs -fs webhdfs://GATEWAYHOST:14000 -cat /user/me/myfile.txt
Hello World!
```

Without the Hadoop client:

You can run all of the standard `hadoop fs` commands by using the WebHDFS REST API and any program that can do `GET`, `PUT`, `POST`, and `DELETE` requests; for example:

```
$ curl "http://GATEWAYHOST:14000/webhdfs/v1/user/me/myfile.txt?op=OPEN&user.name=me"
Hello World!
```

Important: The user.name parameter is valid only if security is disabled. In a secure cluster, you must a initiate a valid Kerberos session.

In general, the command will look like this:

```
$ curl "http://GATEWAYHOST/webhdfs/v1/PATH?[user.name=USER&]op=…"
```

You can find a full explanation of the commands in the WebHDFS REST API documentation.

Submitting and Monitoring Jobs

The Oozie REST API supports the direct submission of jobs for MapReduce, Pig, and Hive; Oozie automatically creates a workflow with a single action. For any other action types, or to execute anything more complicated than a single job, you must create an actual workflow. Required files (JAR files, input data) must already exist on HDFS; if they do not, you can use HttpFS to upload the files.

With the Oozie client:

All of the standard Oozie commands will work. You can find a full explanation of the commands in the documentation for the command-line utilities.

Without the Oozie client:

You can run all of the standard Oozie commands by using the REST API and any program that can do GET, PUT, and POST requests. You can find a full explanation of the commands in the Oozie Web Services API documentation.

How To Set Up Access to Cloudera EDH or Cloudera Director (Microsoft Azure Marketplace)

The Cloudera Enterprise Data Hub (EDH) and Cloudera Director bundles that are available on the Microsoft Azure Marketplace only support SSH access (port 22). To access Cloudera Manager (port 7180), Cloudera Director (port 7189), or other services, you can:

- Set up a SOCKS (Sockets Secure protocol) proxy on your client machine. Cloudera recommends that you use this option.
- Add inbound rules to the Network Security Group in the Azure instance after you deploy EDH or Director to Azure.

Configure the SOCKS Proxy

The SOCKS5 protocol is implemented as a client and server process that enables traversal of IP network firewalls. After you configure the SOCKS proxy, your browser resolves DNS lookups using the Microsoft Azure network (through the proxy server), and lets you connect to services using internal FQDNs or private IP address.

With this approach, you complete the following tasks:

- Set up a single SSH tunnel to one of the hosts on the network and create a SOCKS proxy on the host.
- Change the browser configuration to perform all lookups through the SOCKS proxy host.

Network Prerequisites

Verify the following prerequisites before you connect to your cluster with a SOCKS proxy:

- You must be able to reach the host that you want to proxy to from the public internet or from the network that you're connecting from.
- The host that you proxy to must be on the same network as the Cloudera services that you're connecting to. For example, if you’re using the Cloudera EDH offering, tunnel to the Cloudera Manager host. If you're using the Cloudera Director offering, tunnel to the Cloudera Director host.

Find the Public IP of the Host

Only one VM is created for the Cloudera Director. Use the public IP of that VM.

For the Cloudera EDH offering, use the public IP of the 0th master node VM: [dnsName]-mn0.

Start the SOCKS Proxy

On Linux

To start a SOCKS proxy over SSH, run the following command:

```
ssh -i your-key-file.pem -CND 1080
the_username_you_specified@publicIP_of_VM
```

The command uses the following parameters:

- `-i your-key-file.pem` Specifies the path to the private key needed to SSH to the Cloudera Director server. Omit if using SSH passwords.
- `C` Sets up compression.
- `N` Suppresses any command execution once established.
- `D` Sets up the SOCKS proxy on a port.
- `1080` The port to set the SOCKS proxy locally.

On Windows

Follow the instructions on Microsoft's website.

Configure Google Chrome to Use the Proxy

By default, Google Chrome uses system-wide proxy settings on a per-profile basis. To start Chrome without these settings, open Chrome through the command line and specify the following:

- The SOCKS proxy port. This must be the same port that you used when you started the proxy.
- The profile. This examples below create a new profile.

Use one of the following commands to create a profile and launch a new instance of Chrome that does not conflict with any currently running Chrome instances.

Linux

```
/usr/bin/google-chrome \
--user-data-dir="$HOME/chrome-with-proxy" \
--proxy-server="socks5://localhost:1080"
```

Mac OS X

```
"/Applications/Google Chrome.app/Contents/MacOS/Google Chrome" \
--user-data-dir="$HOME/chrome-with-proxy" \
--proxy-server="socks5://localhost:1080"
```

Microsoft Windows

```
"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" ^
--user-data-dir="%USERPROFILE%\chrome-with-proxy" ^
--proxy-server="socks5://localhost:1080"
```

In this Chrome session, you can use the private IP address or internal FQDN to connect to any host that is accessible by Cloudera Director. For example, if you proxy to the Cloudera Director server, you can connect to Cloudera Director as if it were local by entering `localhost:7189` in the Chrome URL bar.

Network Security Group

Warning: This method is not recommended for any purpose other than a Proof of Concept. If the data is not carefully locked down, it will be accessible to hackers and malicious entities.

On portal.azure.com, find the Network Security Group(s) and add inbound rules for the various services. You may have to create these rules for the services. Refer to Cloudera documentation for more information on ports used by Cloudera Manager, CDH components, managed services, and third-party components.

How to Use Self-Signed Certificates for TLS

Self-signed certificates should not be used for production deployments. Self-signed certificates are created and stored in the keystore specified during the key-generation process, and should be replaced by a signed certificate. Using self-signed certificates requires generating and distributing the certificates and establishing explicit trust for the certificate.

However, using self-signed certificates lets you easily obtain certificates for TLS/SSL configuration and may be appropriate for non-production or test setups. See Data in Transit Encryption (TLS/SSL) on page 179 and Configuring Cloudera Manager Clusters for TLS/SSL on page 183 for more information.

Replace paths, file names, aliases, and other examples in the commands below for your system.

1. Create the directory for the certificates:

```
$ mkdir -p /opt/cloudera/security/x509/ /opt/cloudera/security/jks/
```

   Give Cloudera Manager access to the directory, set the correct permissions, and then change to the directory:

```
$ sudo chown -R cloudera-scm:cloudera-scm /opt/cloudera/security/jks
$ sudo umask 0700
$ cd /opt/cloudera/security/jks
```

2. Generate the key pair and self-signed certificate, storing everything in the keystore with the same password for keystore and storepass, as shown below. Use the FQDN of the current host for the CN to avoid raising a `java.io.IOException: HTTPS hostname wrong` exception. Replace values for OU, O, L, ST, and C with entries appropriate for your environment:

```
keytool -genkeypair -alias cmhost -keyalg RSA -keysize 2048 -dname "cn=cm01.example.com,
 ou=Department,
o=Company, l=City, st=State, c=US" -keypass password -keystore example.jks -storepass
password
```

3. Copy the default Java truststore (`cacerts`) to the alternate system truststore (`jssecacerts`):

```
$ sudo cp $JAVA_HOME/jre/lib/security/cacerts $JAVA_HOME/jre/lib/security/jssecacerts
```

4. Export the certificate from the keystore (`example.jks`).

```
$ keytool -export -alias cmhost -keystore example.jks -rfc -file selfsigned.cer
```

5. Copy the self-signed certificate (`selfsigned.cer`) to the `/opt/cloudera/security/x509/` directory.

```
$ cp selfsigned.cer /opt/cloudera/security/x509/cmhost.pem
```

6. Import the public key into the alternate system truststore (`jssecacerts`), so that any process that runs with Java on this machine will trust the key. The default password for the Java truststore is `changeit`. Do not use the password created for the keystore in step 2.

```
$ keytool -import -alias cmhost -file /opt/cloudera/security/jks/selfsigned.cer
-keystore $JAVA_HOME/jre/lib/security/jssecacerts -storepass changeit
```

   Important: Repeat this process on each host in the cluster.

7. Rename the keystore:

```
$ mv /opt/cloudera/security/jks/example.jks /opt/cloudera/security/jks/cmhost-keystore.jks
```

You can also delete the certificate because it was copied to the appropriate path in step 5.

```
$ rm /opt/cloudera/security/jks/selfsigned.cer
```

The self-signed certificate set up is complete.

Troubleshooting

This section includes troubleshooting for several configurable security related components (Kerberos, TLS/SSL, for example). You will also find some architectural details about the underpinnings of some components, as well as some details about various error codes and messages that may arise.

Troubleshooting Authentication Issues

Typically, if there are problems with security, Hadoop will display generic messages about the cause of the problem. This topic contains solutions to potential problems you might face when configuring a secure cluster:

Common Security Problems and Their Solutions

This troubleshooting section contains sample Kerberos configuration files, `krb5.conf` and `kdc.conf` for your reference. It also has solutions to potential problems you might face when configuring a secure cluster:

Issues with Generate Credentials

Cloudera Manager uses a command called **Generate Credentials** to create the accounts needed by CDH for enabling authentication using Kerberos. The command is triggered automatically when you are using the Kerberos Wizard or making changes to your cluster that will require new Kerberos principals.

When configuring Kerberos, if CDH services do not start, and on the Cloudera Manager **Home** > **Status** tab you see a validation error, `Role is missing Kerberos keytab`, it means the **Generate Credentials** command failed. To see the output of the command, go to the **Home** > **Status** tab and click the **All Recent Commands** tab.

Here are some common error messages:

Problems	Possible Causes	Solutions
With Active Directory		
`ldap_sasl_interactive_bind_s: Can't contact LDAP server (-1)`	The Domain Controller specified is incorrect or LDAPS has not been enabled for it.	Verify the KDC configuration by going to the Cloudera Manager Admin Console and go to **Administration> Settings> Kerberos**. Also check that LDAPS is enabled for Active Directory.
`ldap_add: Insufficient access (50)`	The Active Directory account you are using for Cloudera Manager does not have permissions to create other accounts.	Use the Delegate Control wizard to grant permission to the Cloudera Manager account to create other accounts. You can also login to Active Directory as the Cloudera Manager user to check that it can create other accounts in your Organizational Unit.
With MIT KDC		
`kadmin: Cannot resolve network address for admin server in requested realm while initializing kadmin interface.`	The hostname for the KDC server is incorrect.	Check the kdc field for your default realm in `krb5.conf` and make sure the hostname is correct.

Running any Hadoop command fails after enabling security.

Description:

A user must have a valid Kerberos ticket to interact with a secure Hadoop cluster. Running any Hadoop command (such as `hadoop fs -ls`) will fail if you do not have a valid Kerberos ticket in your credentials cache. If you do not have a valid ticket, you will receive an error such as:

```
11/01/04 12:08:12 WARN ipc.Client: Exception encountered while connecting to the server
 : javax.security.sasl.SaslException:
GSS initiate failed [Caused by GSSException: No valid credentials provided (Mechanism
level: Failed to find any Kerberos tgt)]
Bad connection to FS. command aborted. exception: Call to nn-host/10.0.0.2:8020 failed
 on local exception: java.io.IOException:
javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSException: No valid
 credentials provided (Mechanism level: Failed to find any Kerberos tgt)]
```

Solution:

You can examine the Kerberos tickets currently in your credentials cache by running the `klist` command. You can obtain a ticket by running the `kinit` command and either specifying a keytab file containing credentials, or entering the password for your principal.

Using the UserGroupInformation class to authenticate Oozie

Secured CDH services mainly use Kerberos to authenticate RPC communication. RPCs are one of the primary means of communication between nodes in a Hadoop cluster. For example, RPCs are used by the YARN NodeManager to communicate with the ResourceManager, or by the HDFS client to communicate with the NameNode.

CDH services handle Kerberos authentication by calling the UGI login method, `loginUserFromKeytab()`, once every time the service starts up. Since Kerberos ticket expiration times are typically short, repeated logins are required to keep the application secure. Long-running CDH applications have to be implemented accordingly to accommodate these repeated logins. If an application is only going to communicate with HDFS, YARN, MRv1, and HBase, then you only need to call the `UserGroupInformation.loginUserFromKeytab()` method at startup, before any actual API activity occurs. The HDFS, YARN, MRv1 and HBase services' RPC clients have their own built-in mechanisms to automatically re-login when a keytab's Ticket-Granting Ticket (TGT) expires. Therefore, such applications do not need to include calls to the UGI re-login method because their RPC client layer performs the re-login task for them.

However, some applications may include other service clients that do not involve the generic Hadoop RPC framework, such as Hive or Oozie clients. Such applications must explicitly call the `UserGroupInformation.getLoginUser().checkTGTAndReloginFromKeytab()` method before every attempt to connect with a Hive or Oozie client. This is because these clients do not have the logic required for automatic re-logins.

This is an example of an infinitely polling Oozie client application:

```
// App startup
UserGroupInformation.loginFromKeytab(KEYTAB_PATH, PRINCIPAL_STRING);
OozieClient client = loginUser.doAs(new PrivilegedAction<OozieClient>() {
    public OozieClient run() {
        try {
            returnnew OozieClient(OOZIE_SERVER_URI);
        } catch (Exception e) {
            e.printStackTrace();
            returnnull;
        }
    }
});

while (true && client != null) {
    // Application's long-running loop

    // Every time, complete the TGT check first
    UserGroupInformation loginUser = UserGroupInformation.getLoginUser();
```

```
    loginUser.checkTGTAndReloginFromKeytab();

    // Perform Oozie client work within the context of the login user object
    loginUser.doAs(new PrivilegedAction<Void>() {
        publicVoid run() {
            try {
                List<WorkflowJob> list = client.getJobsInfo("");
                for (WorkflowJob wfJob : list) {
                    System.out.println(wfJob.getId());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        } // End of function block
    }); // End of doAs
} // End of loop
```

Java is unable to read the Kerberos credentials cache created by versions of MIT Kerberos 1.8.1 or higher.

Description:

If you are running MIT Kerberos 1.8.1 or higher, the following error will occur when you attempt to interact with the Hadoop cluster, even after successfully obtaining a Kerberos ticket using `kinit`:

```
11/01/04 12:08:12 WARN ipc.Client: Exception encountered while connecting to the server
 : javax.security.sasl.SaslException:
GSS initiate failed [Caused by GSSException: No valid credentials provided (Mechanism
level: Failed to find any Kerberos tgt)]
Bad connection to FS. command aborted. exception: Call to nn-host/10.0.0.2:8020 failed
 on local exception: java.io.IOException:
javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSException: No valid
 credentials provided (Mechanism level: Failed to find any Kerberos tgt)]
```

Because of a change `[1]` in the format in which MIT Kerberos writes its credentials cache, there is a bug `[2]` in the Oracle JDK 6 Update 26 and earlier that causes Java to be unable to read the Kerberos credentials cache created by versions of MIT Kerberos 1.8.1 or higher. Kerberos 1.8.1 is the default in Ubuntu Lucid and higher releases and Debian Squeeze and higher releases. (On RHEL and CentOS, an older version of MIT Kerberos which does not have this issue, is the default.)

Footnotes:

`[1]` MIT Kerberos change: http://krbdev.mit.edu/rt/Ticket/Display.html?id=6206

`[2]` Report of bug in Oracle JDK 6 Update 26 and lower:
http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6979329

Solution:

If you encounter this problem, you can work around it by running `kinit -R` after running `kinit` initially to obtain credentials. Doing so will cause the ticket to be renewed, and the credentials cache rewritten in a format which Java can read. To illustrate this:

```
$ klist
klist: No credentials cache found (ticket cache FILE:/tmp/krb5cc_1000)
$ hadoop fs -ls
11/01/04 13:15:51 WARN ipc.Client: Exception encountered while connecting to the server
 : javax.security.sasl.SaslException:
GSS initiate failed [Caused by GSSException: No valid credentials provided (Mechanism
level: Failed to find any Kerberos tgt)]
Bad connection to FS. command aborted. exception: Call to nn-host/10.0.0.2:8020 failed
 on local exception: java.io.IOException:
javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSException: No valid
 credentials provided (Mechanism level: Failed to find any Kerberos tgt)]
$ kinit
Password for atm@YOUR-REALM.COM:
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: atm@YOUR-REALM.COM

Valid starting     Expires            Service principal
01/04/11 13:19:31  01/04/11 23:19:31  krbtgt/YOUR-REALM.COM@YOUR-REALM.COM
 renew until 01/05/11 13:19:30
$ hadoop fs -ls
11/01/04 13:15:59 WARN ipc.Client: Exception encountered while connecting to the server
 : javax.security.sasl.SaslException:
GSS initiate failed [Caused by GSSException: No valid credentials provided (Mechanism
level: Failed to find any Kerberos tgt)]
Bad connection to FS. command aborted. exception: Call to nn-host/10.0.0.2:8020 failed
 on local exception: java.io.IOException:
javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSException: No valid
 credentials provided (Mechanism level: Failed to find any Kerberos tgt)]
$ kinit -R
$ hadoop fs -ls
Found 6 items
drwx------   - atm atm          0 2011-01-02 16:16 /user/atm/.staging
```

Note:

This workaround for Problem 2 requires the initial ticket to be renewable. Note that whether or not you can obtain renewable tickets is dependent upon a KDC-wide setting, as well as a per-principal setting for both the principal in question and the Ticket Granting Ticket (TGT) service principal for the realm. A non-renewable ticket will have the same values for its "valid starting" and "renew until" times. If the initial ticket is not renewable, the following error message is displayed when attempting to renew the ticket:

```
kinit: Ticket expired while renewing credentials
```

java.io.IOException: Incorrect permission

Description:

An error such as the following example is displayed if the user running one of the Hadoop daemons has a umask of 0002, instead of 0022:

```
java.io.IOException: Incorrect permission for
/var/folders/B3/B3d2vCm4F+mmWzVPB89W6E+++TI/-Tmp-/tmpYTil84/dfs/data/data1,
expected: rwxr-xr-x, while actual: rwxrwxr-x
       at org.apache.hadoop.util.DiskChecker.checkPermission(DiskChecker.java:107)
       at
org.apache.hadoop.util.DiskChecker.mkdirsWithExistsAndPermissionCheck(DiskChecker.java:144)

       at org.apache.hadoop.util.DiskChecker.checkDir(DiskChecker.java:160)
      at org.apache.hadoop.hdfs.server.datanode.DataNode.makeInstance(DataNode.java:1484)

       at
org.apache.hadoop.hdfs.server.datanode.DataNode.instantiateDataNode(DataNode.java:1432)

       at
org.apache.hadoop.hdfs.server.datanode.DataNode.instantiateDataNode(DataNode.java:1408)

      at org.apache.hadoop.hdfs.MiniDFSCluster.startDataNodes(MiniDFSCluster.java:418)

       at org.apache.hadoop.hdfs.MiniDFSCluster.<init>(MiniDFSCluster.java:279)
       at org.apache.hadoop.hdfs.MiniDFSCluster.<init>(MiniDFSCluster.java:203)
       at
org.apache.hadoop.test.MiniHadoopClusterManager.start(MiniHadoopClusterManager.java:152)

       at
org.apache.hadoop.test.MiniHadoopClusterManager.run(MiniHadoopClusterManager.java:129)
       at
org.apache.hadoop.test.MiniHadoopClusterManager.main(MiniHadoopClusterManager.java:308)
```

```
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)

        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at
org.apache.hadoop.util.ProgramDriver$ProgramDescription.invoke(ProgramDriver.java:68)
        at org.apache.hadoop.util.ProgramDriver.driver(ProgramDriver.java:139)
        at org.apache.hadoop.test.AllTestDriver.main(AllTestDriver.java:83)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)

        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:186)
```

Solution:

Make sure that the `umask` for `hdfs` and `mapred` is `0022`.

A cluster fails to run jobs after security is enabled.

Description:

A cluster that was previously configured to *not* use security may fail to run jobs for certain users on certain TaskTrackers (MRv1) or NodeManagers (YARN) after security is enabled due to the following sequence of events:

1. A cluster is at some point in time configured *without* security enabled.
2. A user X runs some jobs on the cluster, which creates a local user directory on each TaskTracker or NodeManager.
3. Security is enabled on the cluster.
4. User X tries to run jobs on the cluster, and the local user directory on (potentially a subset of) the TaskTrackers or NodeManagers is owned by the wrong user or has overly-permissive permissions.

The bug is that after step 2, the local user directory on the TaskTracker or NodeManager should be cleaned up, but isn't.

If you're encountering this problem, you may see errors in the TaskTracker or NodeManager logs. The following example is for a TaskTracker on MRv1:

```
10/11/03 01:29:55 INFO mapred.JobClient: Task Id : attempt_201011021321_0004_m_000011_0,
 Status : FAILED
Error initializing attempt_201011021321_0004_m_000011_0:
java.io.IOException: org.apache.hadoop.util.Shell$ExitCodeException:
at org.apache.hadoop.mapred.LinuxTaskController.runCommand(LinuxTaskController.java:212)

at
org.apache.hadoop.mapred.LinuxTaskController.initializeUser(LinuxTaskController.java:442)

at
org.apache.hadoop.mapreduce.server.tasktracker.Localizer.initializeUserDirs(Localizer.java:272)

at org.apache.hadoop.mapred.TaskTracker.localizeJob(TaskTracker.java:963)
at org.apache.hadoop.mapred.TaskTracker.startNewTask(TaskTracker.java:2209)
at org.apache.hadoop.mapred.TaskTracker$TaskLauncher.run(TaskTracker.java:2174)
Caused by: org.apache.hadoop.util.Shell$ExitCodeException:
at org.apache.hadoop.util.Shell.runCommand(Shell.java:250)
at org.apache.hadoop.util.Shell.run(Shell.java:177)
at org.apache.hadoop.util.Shell$ShellCommandExecutor.execute(Shell.java:370)
at org.apache.hadoop.mapred.LinuxTaskController.runCommand(LinuxTaskController.java:203)

... 5 more
```

Solution:

Delete the `mapred.local.dir` or `yarn.nodemanager.local-dirs` directories for that user across the cluster.

The NameNode does not start and KrbException Messages (906) and (31) are displayed.

Description:

When you attempt to start the NameNode, a login failure occurs. This failure prevents the NameNode from starting and the following KrbException messages are displayed:

```
Caused by: KrbException: Integrity check on decrypted field failed (31) - PREAUTH_FAILED}}
```

and

```
Caused by: KrbException: Identifier does not match expected value (906)
```

> **Note:**
>
> These KrbException error messages are displayed only if you enable debugging output. See Enabling Debugging Output for the Sun Kerberos Classes.

Solution:

Although there are several possible problems that can cause these two KrbException error messages to display, here are some actions you can take to solve the most likely problems:

- If you are using CentOS/Red Hat Enterprise Linux 5.6 or higher, or Ubuntu, which use AES-256 encryption by default for tickets, you must install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy File on all cluster and Hadoop user machines. For information about how to verify the type of encryption used in your cluster, see Step 3: If you are Using AES-256 Encryption, Install the JCE Policy File on page 90. Alternatively, you can change your `kdc.conf` or `krb5.conf` to not use AES-256 by removing `aes256-cts:normal` from the `supported_enctypes` field of the `kdc.conf` or `krb5.conf` file. Note that after changing the `kdc.conf` file, you'll need to restart both the KDC and the kadmin server for those changes to take affect. You may also need to recreate or change the password of the relevant principals, including potentially the TGT principal (krbtgt/REALM@REALM).
- In the [realms] section of your `kdc.conf` file, in the realm corresponding to `HADOOP.LOCALDOMAIN`, add (or replace if it's already present) the following variable:

```
supported_enctypes = des3-hmac-sha1:normal arcfour-hmac:normal des-hmac-sha1:normal
des-cbc-md5:normal des-cbc-crc:normal des-cbc-crc:v4 des-cbc-crc:afs3
```

- Recreate the `hdfs` keytab file and `mapred` keytab file using the `-norandkey` option in the `xst` command (for details, see Step 4: Create and Deploy the Kerberos Principals and Keytab Files on page 91).

```
kadmin.local: xst -norandkey -k hdfs.keytab hdfs/fully.qualified.domain.name
HTTP/fully.qualified.domain.name
kadmin.local: xst -norandkey -k mapred.keytab mapred/fully.qualified.domain.name
HTTP/fully.qualified.domain.name
```

The NameNode starts but clients cannot connect to it and error message contains enctype code 18.

Description:

The NameNode keytab file does not have an AES256 entry, but client tickets do contain an AES256 entry. The NameNode starts but clients cannot connect to it. The error message does not refer to "AES256", but does contain an enctype code "18".

Solution:

Make sure the "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy File" is installed or remove `aes256-cts:normal` from the `supported_enctypes` field of the `kdc.conf` or `krb5.conf` file. For more information, see the first suggested solution above for Problem 5.

For more information about the Kerberos encryption types, see http://www.iana.org/assignments/kerberos-parameters/kerberos-parameters.xml.

(MRv1 Only) Jobs won't run and TaskTracker is unable to create a local mapred directory.

Description:

The TaskTracker log contains the following error message:

```
11/08/17 14:44:06 INFO mapred.TaskController: main : user is atm
11/08/17 14:44:06 INFO mapred.TaskController: Failed to create directory
/var/log/hadoop/cache/mapred/mapred/local1/taskTracker/atm - No such file or directory
11/08/17 14:44:06 WARN mapred.TaskTracker: Exception while localization
java.io.IOException: Job initialization failed (20)
        at
org.apache.hadoop.mapred.LinuxTaskController.initializeJob(LinuxTaskController.java:191)

        at org.apache.hadoop.mapred.TaskTracker$4.run(TaskTracker.java:1199)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:396)
        at
org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1127)
        at org.apache.hadoop.mapred.TaskTracker.initializeJob(TaskTracker.java:1174)
        at org.apache.hadoop.mapred.TaskTracker.localizeJob(TaskTracker.java:1089)
        at org.apache.hadoop.mapred.TaskTracker.startNewTask(TaskTracker.java:2257)
       at org.apache.hadoop.mapred.TaskTracker$TaskLauncher.run(TaskTracker.java:2221)
Caused by: org.apache.hadoop.util.Shell$ExitCodeException:
        at org.apache.hadoop.util.Shell.runCommand(Shell.java:255)
        at org.apache.hadoop.util.Shell.run(Shell.java:182)
        at org.apache.hadoop.util.Shell$ShellCommandExecutor.execute(Shell.java:375)
        at
org.apache.hadoop.mapred.LinuxTaskController.initializeJob(LinuxTaskController.java:184)

        ... 8 more
```

Solution:

Make sure the value specified for `mapred.local.dir` is identical in `mapred-site.xml` and `taskcontroller.cfg`. If the values are different, the error message above is returned.

(MRv1 Only) Jobs will not run and TaskTracker is unable to create a Hadoop logs directory.

Description:

The TaskTracker log contains an error message similar to the following :

```
11/08/17 14:48:23 INFO mapred.TaskController: Failed to create directory
/home/atm/src/cloudera/hadoop/build/hadoop-0.23.2-cdh3u1-SNAPSHOT/logs1/userlogs/job_201108171441_0004
 - No such file or directory
11/08/17 14:48:23 WARN mapred.TaskTracker: Exception while localization
java.io.IOException: Job initialization failed (255)
        at
org.apache.hadoop.mapred.LinuxTaskController.initializeJob(LinuxTaskController.java:191)

        at org.apache.hadoop.mapred.TaskTracker$4.run(TaskTracker.java:1199)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:396)
        at
org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1127)
        at org.apache.hadoop.mapred.TaskTracker.initializeJob(TaskTracker.java:1174)
        at org.apache.hadoop.mapred.TaskTracker.localizeJob(TaskTracker.java:1089)
```

```
        at org.apache.hadoop.mapred.TaskTracker.startNewTask(TaskTracker.java:2257)
       at org.apache.hadoop.mapred.TaskTracker$TaskLauncher.run(TaskTracker.java:2221)
Caused by: org.apache.hadoop.util.Shell$ExitCodeException:
        at org.apache.hadoop.util.Shell.runCommand(Shell.java:255)
        at org.apache.hadoop.util.Shell.run(Shell.java:182)
        at org.apache.hadoop.util.Shell$ShellCommandExecutor.execute(Shell.java:375)
        at
org.apache.hadoop.mapred.LinuxTaskController.initializeJob(LinuxTaskController.java:184)

        ... 8 more
```

Solution:

In MRv1, the default value specified for `hadoop.log.dir` in `mapred-site.xml` is `/var/log/hadoop-0.20-mapreduce`. The path must be owned and be writable by the `mapred` user. If you change the default value specified for `hadoop.log.dir`, make sure the value is identical in `mapred-site.xml` and `taskcontroller.cfg`. If the values are different, the error message above is returned.

After you enable cross-realm trust, you can run Hadoop commands in the local realm but not in the remote realm.

Description:

After you enable cross-realm trust, authenticating as a principal in the local realm will allow you to successfully run Hadoop commands, but authenticating as a principal in the remote realm will not allow you to run Hadoop commands. The most common cause of this problem is that the principals in the two realms either do not have the same encryption type, or the cross-realm principals in the two realms do not have the same password. This issue manifests itself because you are able to get Ticket Granting Tickets (TGTs) from both the local and remote realms, but you are unable to get a service ticket to allow the principals in the local and remote realms to communicate with each other.

Solution:

On the local MIT KDC server host, type the following command in the kadmin.local or kadmin shell to add the cross-realm krbtgt principal:

```
kadmin:  addprinc -e "<enc_type_list>"
krbtgt/YOUR-LOCAL-REALM.COMPANY.COM@AD-REALM.COMPANY.COM
```

where the `<enc_type_list>` parameter specifies the types of encryption this cross-realm krbtgt principal will support: AES, DES, or RC4 encryption. You can specify multiple encryption types using the parameter in the command above, what's important is that at least one of the encryption types parameters corresponds to the encryption type found in the tickets granted by the KDC in the remote realm. For example:

```
kadmin:  addprinc -e "aes256-cts:normal rc4-hmac:normal des3-hmac-sha1:normal"
krbtgt/YOUR-LOCAL-REALM.COMPANY.COM@AD-REALM.COMPANY.COM
```

(MRv1 Only) Jobs won't run and cannot access files in mapred.local.dir

Description:

The TaskTracker log contains the following error message:

```
WARN org.apache.hadoop.mapred.TaskTracker: Exception while localization
java.io.IOException: Job initialization failed (1)
```

Solution:

1. Add the `mapred` user to the `mapred` and hadoop groups on all hosts.
2. Restart all TaskTrackers.

Users are unable to obtain credentials when running Hadoop jobs or commands.

Description:

This error occurs because the ticket message is too large for the default UDP protocol. An error message similar to the following may be displayed:

```
13/01/15 17:44:48 DEBUG ipc.Client: Exception encountered while connecting to the server
 : javax.security.sasl.SaslException:
GSS initiate failed [Caused by GSSException: No valid credentials provided (Mechanism
level: Fail to create credential.
(63) - No service creds)]
```

Solution:

Force Kerberos to use TCP instead of UDP by adding the following parameter to `libdefaults` in the `krb5.conf` file on the client(s) where the problem is occurring.

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

If you choose to manage `krb5.conf` through Cloudera Manager, this will automatically get added to `krb5.conf`.

> **Note:**
>
> When sending a message to the KDC, the library will try using TCP before UDP if the size of the ticket message is larger than the setting specified for the `udp_preference_limit` property. If the ticket message is smaller than `udp_preference_limit` setting, then UDP will be tried before TCP. Regardless of the size, both protocols will be tried if the first attempt fails.

`Request is a replay` exceptions in the logs.

Description:

The root cause of this exception is that Kerberos uses a second-resolution timestamp to protect against replay attacks (where an attacker can record network traffic, and play back recorded requests later to gain elevated privileges). That is, incoming requests are cached by Kerberos for a little while, and if there are similar requests within a few seconds, Kerberos will be able to detect them as replay attack attempts. However, if there are multiple valid Kerberos requests coming in at the same time, these may also be misjudged as attacks for the following reasons:

- **Multiple services in the cluster are using the same Kerberos principal.** All secure clients that run on multiple machines should use unique Kerberos principals for each machine. For example, rather than connecting as a service principal `myservice@EXAMPLE.COM`, services should have per-host principals such as `myservice/host123.example.com@EXAMPLE.COM`.
- **Clocks not in sync:** All hosts should run NTP so that clocks are kept in sync between clients and servers.

While having different principals for each service, and clocks in sync helps mitigate the issue, there are, however, cases where even if all of the above are implemented, the problem still persists. In such a case, disabling the cache (*and the replay protection as a consequence*), will allow parallel requests to succeed. This compromise between usability and security can be applied by setting the KRB5RCACHETYPE environment variable to none.

Note that the KRB5RCACHETYPE is not automatically detected by Java applications. For Java-based components:

- Ensure that the cluster runs on JDK 8.
- To disable the replay cache, add `-Dsun.security.krb5.rcache=none` to the Java Opts/Arguments of the targeted JVM. For example, HiveServer2 or the Sentry service.

For more information, refer the MIT KDC documentation.

Symptom: The following exception shows up in the logs for one or more of the Hadoop daemons:

```
2013-02-28 22:49:03,152 INFO  ipc.Server (Server.java:doRead(571)) - IPC Server listener
 on 8020: readAndProcess threw exception javax.security.sasl.SaslException: GSS initiate
 failed [Caused by GSSException: Failure unspecified at GSS-API level (Mechanism l
javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSException: Failure
 unspecified at GSS-API level (Mechanism level: Request is a replay (34))]
        at
com.sun.security.sasl.gsskerb.GssKrb5Server.evaluateResponse(GssKrb5Server.java:159)
       at org.apache.hadoop.ipc.Server$Connection.saslReadAndProcess(Server.java:1040)

        at org.apache.hadoop.ipc.Server$Connection.readAndProcess(Server.java:1213)
        at org.apache.hadoop.ipc.Server$Listener.doRead(Server.java:566)
        at org.apache.hadoop.ipc.Server$Listener$Reader.run(Server.java:363)
Caused by: GSSException: Failure unspecified at GSS-API level (Mechanism level: Request
 is a replay (34))
        at sun.security.jgss.krb5.Krb5Context.acceptSecContext(Krb5Context.java:741)
        at sun.security.jgss.GSSContextImpl.acceptSecContext(GSSContextImpl.java:323)
        at sun.security.jgss.GSSContextImpl.acceptSecContext(GSSContextImpl.java:267)
        at
com.sun.security.sasl.gsskerb.GssKrb5Server.evaluateResponse(GssKrb5Server.java:137)
        ... 4 more
Caused by: KrbException: Request is a replay (34)
        at sun.security.krb5.KrbApReq.authenticate(KrbApReq.java:300)
        at sun.security.krb5.KrbApReq.<init>(KrbApReq.java:134)
       at sun.security.jgss.krb5.InitSecContextToken.<init>(InitSecContextToken.java:79)

        at sun.security.jgss.krb5.Krb5Context.acceptSecContext(Krb5Context.java:724)
        ... 7 more
```

In addition, this problem can manifest itself as performance issues for all clients in the cluster, including dropped connections, timeouts attempting to make RPC calls, and so on.

Cloudera Manager cluster services fail to start

Possible Causes and Solutions:

- Check that the encryption types are matched between your KDC and `krb5.conf` on all hosts.

  Solution: If you are using AES-256, follow the instructions at [Step 2: If You are Using AES-256 Encryption, Install the JCE Policy File](#) on page 52 to deploy the JCE policy file on all hosts.

- If the version of the JCE policy files does not match the version of Java installed on a node, then services will not start. This is because the cryptographic signatures of the JCE policy files cannot be verified if the wrong version is installed. For example, if a DataNode does not start, you will see the following error in the logs to show that verification of the cryptographic signature within the JCE policy files failed.

```
Exception in secureMain
java.lang.ExceptionInInitializerError
at javax.crypto.KeyGenerator.nextSpi(KeyGenerator.java:324)
at javax.crypto.KeyGenerator.<init>(KeyGenerator.java:157)
.
.
.
Caused by: java.lang.SecurityException: The jurisdiction policy files are not signed by
 a trusted signer!
at javax.crypto.JarVerifier.verifyPolicySigned(JarVerifier.java:289)
at javax.crypto.JceSecurity.loadPolicies(JceSecurity.java:316)
at javax.crypto.JceSecurity.setupJurisdictionPolicies(JceSecurity.java:261)
...
```

Solution: Download the correct JCE policy files for the version of Java you are running:

- Java 6
- Java 7

Download and unpack the zip file. Copy the two JAR files to the `$JAVA_HOME/jre/lib/security` directory on each node within the cluster.

HDFS Encryption Issues

This topic contains HDFS Encryption-specific troubleshooting information in the form of issues you might face when encrypting HDFS files/directories and their workarounds.

Retrieval of encryption keys fails

Description

You see the following error when trying to list encryption keys

```
user1@example-sles-4:~> hadoop key list
Cannot list keys for KeyProvider: KMSClientProvider[https:
//example-sles-2.example.com:16000/kms/v1/]: Retrieval of all keys failed.
```

Solution

Make sure your truststore has been updated with the relevant certificate(s), such as the Key Trustee server certificate.

DistCp between unencrypted and encrypted locations fails

Description

By default, DistCp compares checksums provided by the filesystem to verify that data was successfully copied to the destination. However, when copying between unencrypted and encrypted locations, the filesystem checksums will not match since the underlying block data is different.

Solution

Specify the `-skipcrccheck` and `-update` distcp flags to avoid verifying checksums.

(CDH 5.6 and lower) Cannot move encrypted files to trash

> **Note:** Starting with CDH 5.7, you can delete files or directories that are part of an HDFS encryption zone. For details, see Trash Behavior with HDFS Transparent Encryption Enabled.

Description

In CDH 5.6 and lower, with HDFS encryption enabled, you cannot move encrypted files or directories to the trash directory.

Solution

To remove encrypted files/directories, use the following command with the `-skipTrash` flag specified to bypass trash.

```
rm -r -skipTrash /testdir
```

NameNode - KMS communication fails after long periods of inactivity

Description

Encrypted files and encryption zones cannot be created if a long period of time (by default, 20 hours) has passed since the last time the KMS and NameNode communicated.

Solution

> **Important:** Upgrading your cluster to the latest CDH 5 release will fix this problem. For instructions, see Upgrading from an Earlier CDH 5 Release to the Latest Release.

For lower CDH 5 releases, there are two possible workarounds to this issue :

- You can increase the KMS authentication token validity period to a very high number. Since the default value is 10 hours, this bug will only be encountered after 20 hours of no communication between the NameNode and the KMS. Add the following property to the `kms-site.xml`Safety Valve:

```
<property>
<name>hadoop.kms.authentication.token.validity</name>
<value>SOME VERY HIGH NUMBER</value>
</property>
```

- You can switch the KMS signature secret provider to the string secret provider by adding the following property to the `kms-site.xml` Safety Valve:

```
<property>
<name>hadoop.kms.authentication.signature.secret</name>
<value>SOME VERY SECRET STRING</value>
</property>
```

HDFS Trash Behaviour with Transparent Encryption Enabled

The Hadoop trash feature helps prevent accidental deletion of files and directories. When you delete a file in HDFS, the file is not immediately expelled from HDFS. Deleted files are first moved to the `/user/<username>/.Trash/Current` directory, with their original filesystem path being preserved. After a user-configurable period of time (`fs.trash.interval`), a process known as trash checkpointing renames the `Current` directory to the current timestamp, that is, `/user/<username>/.Trash/<timestamp>`. The checkpointing process also checks the rest of the `.Trash` directory for any existing timestamp directories and removes them from HDFS permanently. You can restore files and directories in the trash simply by moving them to a location outside the `.Trash` directory.

Trash Behaviour with HDFS Transparent Encryption Enabled

Starting with CDH 5.7, you can delete files or directories that are part of an HDFS encryption zone. As is evident from the procedure described above, moving and renaming files or directories is an important part of trash handling in HDFS. However, currently HDFS transparent encryption only supports renames *within* an encryption zone. To accommodate this, HDFS creates a local `.Trash` directory every time a new encryption zone is created. For example, when you create an encryption zone, enc_zone, HDFS will also create the `/enc_zone/.Trash/` subdirectory. Files deleted from enc_zone are moved to `/enc_zone/.Trash/<username>/Current/`. After the checkpoint, the `Current` directory is renamed to the current timestamp, `/enc_zone/.Trash/<username>/<timestamp>`.

If you delete the entire encryption zone, it will be moved to the `.Trash` directory under the user's home directory, `/users/<username>/.Trash/Current/enc_zone`. Trash checkpointing will occur only after the entire zone has been moved to `/users/<username>/.Trash`. However, if the user's home directory is already part of an encryption zone, then attempting to delete an encryption zone will fail because you cannot move or rename directories across encryption zones.

Troubleshooting Kerberos Issues

This topic describes the steps you can take to investigate problems with Kerberos authentication. It contains some sample KDC configuration scripts that you can use to make sure your cluster was configured correctly. The following sections also have instructions on using the Kerberos command-line tools, `kinit` and `klist`, to investigate the KDC and cluster setup. Finally, you can use the instructions described below to enable debugging for Kerberos using either the command-line or Cloudera Manager.

Verifying Kerberos Configuration

When you're faced with a Kerberos-related issue, first try to pinpoint the cause of failure. A Kerberized deployment has several potential points of failure. These include the KDC itself, missing Kerberos or OS packages, incorrect mapping of Kerberos realms, among others. For example, you could start by investigating whether the issue is with a user with faulty credentials, or with the service that is failing to authenticate users. Another good starting point is to make sure that the Kerberos configuration files have been configured correctly and are being deployed consistently across all cluster hosts.

If the issue you are diagnosing is not already obvious to you, Cloudera recommends you begin with auditing your Kerberos deployment. Use this audit to confirm that you have followed the Kerberos steps as listed in the Cloudera Security Guide, and that your cluster has been configured correctly. Make sure you perform the following configuration checks:

- Confirm that your `/etc/hosts` file conforms to Cloudera Manager's installation requirements. Verify forward and reverse name resolution for all cluster hosts, including the KDC hosts, MIT or AD.
- Ensure the required Kerberos server and workstation packages based on the version of the OS you are using.
- Check whether the `hadoop.security.auth_to_local` property in `core-site.xml` has the proper mappings for *all* trusted Kerberos realms, especially the HDFS trusted realms. Do this for every service that is using Kerberos.
- Verify your Kerberos configuration using the sample `krb5.conf` and `kdc.conf` files provided below.
- Review the configuration of all the KDC, `REALM`, and domain hosts referenced in the `krb5.conf` and `kdc.conf` files. The KDC host in particular, is a common point-of-failure and you may have to begin troubleshooting there. Ensure that the `REALM` set in `krb5.conf` has the correct hostname listed for the KDC. If you are using cross-realm authentication, see Reviewing Service Ticket Credentials in Cross Realm Deployments on page 508.
- Check whether the services using Kerberos are running and responding properly with `kinit/klist`.
- Attempt to authenticate to Cloudera Manager using cluster service credentials specific to the issue or affected service. Examine the issued credentials if you are able to successfully authenticate with the service keytab.
- Use `klist` to list the principals present within a service keytab to ensure each service has one.
- Enabling debugging using either the command line or Cloudera Manager.

Sample Kerberos Configuration Files

/etc/krb5.conf

The `/etc/krb5.conf` file is the configuration a client uses to access a realm through its configured KDC. The `krb5.conf` maps the realm to the available servers supporting those realms. It also defines the host-specific configuration rules for how tickets are requested and granted.

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 default_realm = EXAMPLE.COM
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
# udp_preference_limit = 1

# set udp_preference_limit = 1 when TCP only should be
# used. Consider using in complex network environments when
# troubleshooting or when dealing with inconsistent
# client behavior or GSS (63) messages.

# uncomment the following if AD cross realm auth is ONLY providing DES encrypted tickets
# allow-weak-crypto = true

[realms]
 AD-REALM.EXAMPLE.COM = {
```

```
  kdc = AD1.ad-realm.example.com:88
  kdc = AD2.ad-realm.example.com:88
  admin_server = AD1.ad-realm.example.com:749
  admin_server = AD2.ad-realm.example.com:749
  default_domain = ad-realm.example.com
 }
 EXAMPLE.COM = {
  kdc = kdc1.example.com:88
  admin_server = kdc1.example.com:749
  default_domain = example.com
 }

# The domain_realm is critical for mapping your host domain names to the kerberos realms
# that are servicing them. Make sure the lowercase left hand portion indicates any
domains or subdomains
# that will be related to the kerberos REALM on the right hand side of the expression.
 REALMs will
# always be UPPERCASE. For example, if your actual DNS domain was test.com but your
kerberos REALM is
# EXAMPLE.COM then you would have,

[domain_realm]
test.com = EXAMPLE.COM
#AD domains and realms are usually the same
ad-domain.example.com = AD-REALM.EXAMPLE.COM
ad-realm.example.com = AD-REALM.EXAMPLE.COM
```

/var/kerberos/krb5kdc

The `kdc.conf` file only needs to be configured on the actual cluster-dedicated KDC, and should be located at `/var/kerberos/krb5kdc`. Only primary and secondary KDCs need access to this configuration file. The contents of this file establish the configuration rules which are enforced for all client hosts in the `REALM`.

```
[kdcdefaults]
 kdc_ports = 88
 kdc_tcp_ports = 88

[realms]
  EXAMPLE.COM = {
  #master_key_type = aes256-cts
  max_renewable_life = 7d 0h 0m 0s
  acl_file = /var/kerberos/krb5kdc/kadm5.acl
  dict_file = /usr/share/dict/words
  admin_keytab = /var/kerberos/krb5kdc/kadm5.keytab
# note that aes256 is ONLY supported in Active Directory in a domain / forrest operating
 at a 2008 or greater functional level.
# aes256 requires that you download and deploy the JCE Policy files for your JDK release
 level to provide
# strong java encryption extension levels like AES256. Make sure to match based on the
 encryption configured within AD for
# cross realm auth, note that RC4 = arcfour when comparing windows and linux enctypes
  supported_enctypes = aes256-cts:normal aes128-cts:normal arcfour-hmac:normal
  default_principal_flags = +renewable, +forwardable
 }
```

kadm5.acl

```
*/admin@HADOOP.COM *
cloudera-scm@HADOOP.COM * flume/*@HADOOP.COM
cloudera-scm@HADOOP.COM * hbase/*@HADOOP.COM
cloudera-scm@HADOOP.COM * hdfs/*@HADOOP.COM
cloudera-scm@HADOOP.COM * hive/*@HADOOP.COM
cloudera-scm@HADOOP.COM * httpfs/*@HADOOP.COM
cloudera-scm@HADOOP.COM * HTTP/*@HADOOP.COM
cloudera-scm@HADOOP.COM * hue/*@HADOOP.COM
cloudera-scm@HADOOP.COM * impala/*@HADOOP.COM
cloudera-scm@HADOOP.COM * mapred/*@HADOOP.COM
cloudera-scm@HADOOP.COM * oozie/*@HADOOP.COM
cloudera-scm@HADOOP.COM * solr/*@HADOOP.COM
```

```
cloudera-scm@HADOOP.COM * sqoop/*@HADOOP.COM
cloudera-scm@HADOOP.COM * yarn/*@HADOOP.COM
cloudera-scm@HADOOP.COM * zookeeper/*@HADOOP.COM
```

Authenticate to Kerberos using the `kinit` command line tool

The `kinit` command line tool is used to authenticate a user, service, system, or device to a KDC. The most basic example is a user authenticating to Kerberos with a username (principal) and password. In the following example, the first attempt uses a wrong password, followed by a second successful attempt.

```
[alice@host1 ~]$ kinit alice@TEST.ORG.LAB
Password for alice@TEST.ORG.LAB: (wrong password)
kinit: Preauthentication failed while getting initial credentials

[alice@host1 ~]$ kinit alice@TEST.ORG.LAB
Password for alice@TEST.ORG.LAB: (correct password)
(note silent return on successful auth)
[alice@host1 ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_10001
Default principal: alice@TEST.ORG.LAB

Valid starting     Expires            Service principal
03/11/14 11:55:39  03/11/14 21:54:55  krbtgt/TEST.ORG.LAB@TEST.ORG.LAB
renew until 03/18/14 11:55:39
```

Another method of authentication is using keytabs with the `kinit` command. You can verify whether authentication was successful by using the `klist` command to show the credentials issued by the KDC. The following example attempts to authenticate the `hdfs` service to the KDC by using the `hdfs` keytab file.

```
[root@host1 312-hdfs-DATANODE]# kinit -kt hdfs.keytab hdfs/host1.test.lab@TEST.LAB
[root@host1 312-hdfs-DATANODE]# klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: hdfs/host1.test.lab@TEST.LAB

Valid starting      Expires      Service principal
03/11/14 11:18:34 03/12/14 11:18:34 krbtgt/TEST.LAB@TEST.LAB
renew until 03/18/14 11:18:34
```

Troubleshooting using service keytabs maintained by Cloudera Manager

Every service managed by Cloudera Manager has a keytab file that is provided at startup by the Cloudera Manager Agent. The most recent keytab files can be examined by navigating to the path, `/var/run/cloudera-scm-agent/process`, with an `ls -ltr` command.

As you can see in the example below, Cloudera Manager service directory names have the form: `###-service-ROLE`. Therefore, if you are troubleshooting the HDFS service, the service directory may be called, `326-hdfs-NAMENODE`.

```
[root@cehd1 ~]# cd /var/run/cloudera-scm-agent/process/
[root@cehd1 process]# ls -ltr | grep NAMENODE | tail -3
drwxr-x--x 3 hdfs         hdfs         4096 Mar  3 23:43 313-hdfs-NAMENODE
drwxr-x--x 3 hdfs         hdfs         4096 Mar  4 00:07 326-hdfs-NAMENODE
drwxr-x--x 3 hdfs         hdfs         4096 Mar  4 00:07 328-hdfs-NAMENODE-nnRpcWait

[root@cehd1 process]# cd 326-hdfs-NAMENODE

[root@cehd1 326-hdfs-NAMENODE]# ls
cloudera_manager_agent_fencer.py              dfs_hosts_allow.txt         hdfs.keytab
                log4j.properties             topology.py
cloudera_manager_agent_fencer_secret_key.txt  dfs_hosts_exclude.txt      hdfs-site.xml
               logs
cloudera-monitor.properties                   event-filter-rules.json
http-auth-signature-secret  navigator.client.properties
core-site.xml                                 hadoop-metrics2.properties  krb5cc_494
                topology.map
```

If you have root access to the `/var/run/cloudera-scm-agent/process` path, you can use any service's keytab file to log in as root or a sudo user to verify whether basic Kerberos authentication is working.

Once you have located a service keytab file, examine its contents using the `klist` command (more on this, later). The `klist` command can show you the credentials stored in a keytab file. For example, to list the credentials stored in the `hdfs.keytab` file, use the following command:

```
[root@host1 326-hdfs-DATANODE]# klist -kt hdfs.keytab

Keytab name: WRFILE:hdfs.keytab
KVNO Timestamp         Principal
---- ----------------- --------------------------------------------------------
   4 02/17/14 19:09:17 HTTP/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 HTTP/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 HTTP/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 HTTP/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 HTTP/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 HTTP/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 hdfs/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 hdfs/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 hdfs/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 hdfs/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 hdfs/host1.test.lab@TEST.LAB
   4 02/17/14 19:09:17 hdfs/host1.test.lab@TEST.LAB
```

Now, attempt to authenticate using the keytab file and a principal within it. In this case, we use the `hdfs.keytab` file with the `hdfs/host1.test.lab@TEST.LAB` principal. Then use the `klist` command without any arguments to see the current user session's credentials.

```
root@host1 312-hdfs-DATANODE]# kinit -kt hdfs.keytab hdfs/host1.test.lab@TEST.LAB
[root@host1 312-hdfs-DATANODE]# klist

Ticket cache: FILE:/tmp/krb5cc_0
Default principal: hdfs/host1.test.lab@TEST.LAB

Valid starting     Expires            Service principal
03/11/14 11:18:34  03/12/14 11:18:34  krbtgt/TEST.LAB@TEST.LAB
renew until 03/18/14 11:18:34
```

Note that Kerberos credentials have an expiry date and time. This means, to make sure Kerberos credentials are valid uniformly over a cluster, all hosts and clients within the cluster should be using NTP and must never drift more than 5 minutes apart from each other. Kerberos session tickets have a limited lifespan, but can be renewed (as indicated in the sample `krb5.conf` and `kdc.conf`). CDH requires renewable tickets for cluster principals. Check whether renewable tickets have been enabled by using a klist command with the `-e` (list key encryption types) and `-f` (list flags set) switches when examining Kerberos sessions and credentials.

Examining Kerberos credentials with `klist`

So far we've only seen basic usage examples of the `klist` command to list the contents of a keytab file, or to examine a user's credentials. To get more information from the `klist` command, such as the encryption types being negotiated, or the flags being set for credentials being issued by the KDC, use the `klist -ef` command. The output for this command will show you the negotiated encryption types for a user or service principal. This is useful information because you may troubleshoot errors caused (especially in cross-realm trust deployments) because an AD or MIT KDC server may not support a particular encryption type. Look for the encryption types under the "Etype" section of the output.

Flags indicate options supported by Kerberos that extend the features of a set of issued credentials. As discussed previously, CDH requires renewable as well as forwardable tickets for successful authentication, especially in cross realm environments. Look for these settings in the "Flags:" section of the `klist -ef` output shown below where, F = Forwardable, and, R = renewable.

For example, if you use the `klist -ef` command in an ongoing user session:

```
[alice@host1 ~]$ klist -ef
Ticket cache: FILE:/tmp/krb5cc_10001
Default principal: alice@TEST.ORG.LAB
Valid starting Expires Service principal
03/11/14 11:55:39 03/11/14 21:54:55 krbtgt/TEST.ORG.LAB@TEST.ORG.LAB
renew until 03/18/14 11:55:39, Flags: FRIA
Etype (skey, tkt): aes256-cts-hmac-sha1-96,
    aes256-cts-hmac-sha1-96
```

Reviewing Service Ticket Credentials in Cross Realm Deployments

When you examine your cluster configuration, make sure you haven't violated any of following the integration rules:

- When negotiating encryption types, follow the realm with the most specific limitations on supported encryption types.
- All realms should be known to one another through the `/etc/krb5.conf` file deployed on the cluster.
- When you make configuration decisions for Active Directory environments, you must evaluate the Domain Functional Level or Forrest Functional Level that is present.

Kerberos will typically negotiate the strongest form of encryption possible between a client and server for authentication into the realm. However, the encryption types for TGTs may sometimes end up being negotiated downward towards the weaker encryption types, which is not desirable. To investigate such issues, check the `kvno` of the cross-realm trust principal (`krbtgt`) as described in the following steps. Replace `CLUSTER.REALM` and `AD.REALM` (or `MIT.REALM`) with the appropriate values for your configured realm. This scenario assumes cross-realm authentication with Active Directory.

1. Once trust has been configured (see sample files in previous section), `kinit` as a system user by authenticating to the AD Kerberos realm.
2. From the command line, perform a `kvno` check of the local and cross-realm `krbtgt` entry. The local representation of this special `REALM` service principal is in the form, `krbtgt/CLUSTER.REALM@CLUSTER.REALM`. The cross-realm principal is named after the trusted realm in the form, `krbtgt/AD.REALM`.

   If the `kvno` check fails, this means cross-realm trust was not set up correctly. Once again review the encryption types in use to make sure there are no incompatibilities or unsupported encryption types being used across realms.

Enabling Debugging in Cloudera Manager for CDH Services

The following instructions are specific to a Cloudera Manager managed-HDFS service and must be modified based on the Kerberized service you are troubleshooting.

1. Go to the Cloudera Manager Admin Console and navigate to the HDFS service.
2. Click **Configuration**.
3. Search for properties specific to the different role types for which you want to enable debugging. For example, if you want to enable debugging for the HDFS NameNode, search for the **NameNode Logging Threshold** property and select at least `DEBUG` level logging.
4. Enable Kerberos debugging by using the HDFS service's Advanced Configuration Snippet. Once again, this may be different for each specific role type or service. For the HDFS NameNode, add the following properties to the HDFS Service Environment Safety Valve:

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

5. Click **Save Changes**.
6. Restart the HDFS service.

The output will be seen in the process logs: `stdout.log` and `stderr.log`. These can be found in the runtime path of the instance: `/var/run/cloudera-scm-agent/process/###-service-ROLE`. Once Cloudera Manager services have been restarted, the most recent instance of the `###-service-ROLE` directory will have debug logs. Use `ls -ltr` in the `/var/run/cloudera-scm-agent/process` path to determine the most current path.

Enabling Debugging for Command Line Troubleshooting

Set the following properties in your environment to produce detailed debugging output of the Kerberos authentication process.

```
# export HADOOP_ROOT_LOGGER=TRACE,console; export HADOOP_JAAS_DEBUG=true; export
HADOOP_OPTS="-Dsun.security.krb5.debug=true"
```

You can then use the following command to copy the console output to the user (with the debugging), along with all output from STDOUT and STDERR to a file.

```
# hadoop fs -ls / > >(tee fsls-logfile.txt) 2>&1
```

Troubleshooting TLS/SSL Issues in Cloudera Manager

To diagnose and resolve issues on a TLS-enabled cluster, you can check all your configuration steps for the cluster. See the appropriate configuration information for your cluster:

- How to Configure TLS for Cloudera Manager uses an internal-CA-signed certificate as an example and steps through all steps for configuring the cluster through Cloudera Level 3 TLS.
- Configuring Cloudera Manager Clusters for TLS/SSL on page 183 uses public-CA signed certificates as an example and provides steps that build on each of the levels.

After checking your settings and finding no obvious misconfiguration, you can take some of the actions detailed below.

Test Connectivity with OpenSSL

From the host that has connectivity issues, run `openssl` as shown below. You can also check that the certificate used by the host is recognized by a trusted CA during the TLS/SSL negotiation.

To check the connection:

```
$ openssl s_client -connect [host.fqdn.name]:[port]
```

For example:

```
$ openssl s_client -connect test1.sec.cloudera.com:7183
```

A return code `0` means `openssl` was able to follow trust server chain of trust through its library of trusted public CAs.

For certificates signed by your organization's internal CA or self-signed certificates, you may need to add the certificate to the truststore using the `openssl` command. Use the `-CAfile` option to specify the path to the root CA so `openssl` can verify the self-signed or internal-CA-signed certificate as follows:

```
$ openssl s_client -connect test1.sec.cloudera.com:7183 -CAfile \
/opt/cloudera/security/CAcerts/RootCA.pem
```

Only the Root CA certificate is needed to establish trust for this test. The result from the command is successful when you see the return code `0` as follows:

```
...
 Verify return code: 0 (ok)
---
```

By default, Cloudera Manager Server writes logs to the `/etc/cloudera-scm-server/cloudera-scm-server.log` file on startup. Successful server start-up using the certificate looks similar to the following log example:

```
2014-10-06 21:33:47,515 INFO WebServerImpl:org.mortbay.log: jetty-6.1.26.cloudera.2
2014-10-06 21:33:47,572 INFO WebServerImpl:org.mortbay.log: Started
SslSelectChannelConnector@0.0.0.0:7183
2014-10-06 21:33:47,573 INFO WebServerImpl:org.mortbay.log: Started
SelectChannelConnector@0.0.0.0:7180
2014-10-06 21:33:47,573 INFO WebServerImpl:com.cloudera.server.cmf.WebServerImpl: Started
 Jetty server.
```

Upload Diagnostic Bundles to Cloudera Support

By default, Cloudera Manager uploads diagnostic bundles over HTTPS to the Cloudera Support server at `cops.cloudera.com`. However, the upload can fail if the Cloudera Manager truststore cannot verify the authenticity of the Cloudera Support server certificate, and that verification process can fail due to Cloudera Manager truststore configuration issues.

To ensure the Cloudera Manager Server truststore contains the public CAs needed to verify Cloudera Support's certificate, you can explicitly establish trust by importing Cloudera Support's certificate into Cloudera Manager's truststore.

> **Note:** Cloudera Support servers use certificates signed by a commercial CA, so this step is typically not needed, unless the default truststore has been altered. Before downloading or adding any certificates, test the connection and verify that the certificate is the source of the connection issue.

Importing Cloudera Support's Certificate into the Cloudera Manager Server Truststore

To obtain Cloudera's public key certificate from the Cloudera Support server:

```
$ openssl s_client -connect cops.cloudera.com:443 | openssl x509 -text -out
/path/to/cloudera-cert.pem
```

To import this certificate into the Cloudera Manager truststore (use paths for your own system):

```
$ keytool -import -keystore /path/to/cm/truststore.jks -file /path/to/cloudera-cert.pem
```

After importing the certificate, confirm that Cloudera Manager is configured for this truststore file, as detailed in Configuring Cloudera Manager Truststore Properties on page 510.

> **Note:** Alternatively, you can use the default Java truststore for your Cloudera Manager cluster deployment, as described in Generate the Cloudera Manager Server Certificate on page 457.

Configuring Cloudera Manager Truststore Properties

After installing the Cloudera Support server certificate into the Cloudera Manager truststore, you must configure Cloudera Manager to use the truststore, as follows:

1. Log into the Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Click the **Security** category.
4. Enter the path to the truststore and the password (if necessary):

Setting	Description
Cloudera Manager TLS/SSL Certificate Trust Store File	Enter the complete Cloudera Manager Server host filesystem path to the truststore (the trust `.jks`). Cloudera Manager Server invokes JVM with `-Djavax.net.ssl.trustStore` to access the specified truststore.

Setting	Description
Cloudera Manager TLS/SSL Certificate Trust Store Password	Specify the password (if there is one) for the truststore file. Password is not required to access the truststore, so you can typically leave this field blank. Cloudera Manager Server invokes JVM with `-Djavax.net.ssl.trustStore.password` if this field has an entry.

5. Click **Save Changes** to save the settings.

Note: See Oracle's JSSE Reference Guide for more information about the JSSE trust mechanism.

YARN, MRv1, and Linux OS Security

Several subsystems are fundamental to Hadoop clusters, specifically, the Jsvc, Task Controller, and Container Executor Programs, documented below.

MRv1 and YARN: The jsvc Program

A set of libraries designed to make Java applications run better on Linux, the `jsvc` program is part of the `bigtop-jsvc` package and installed in either `/usr/lib/bigtop-utils/jsvc` or `/usr/libexec/bigtop-utils/jsvc` depending on the particular Linux flavor.

In particular, `jsvc` (more info) is used to start the DataNode listening on low port numbers. Its entry point is the `SecureDataNodeStarter` class, which implements the Daemon interface that `jsvc` expects. `jsvc` is run as root, and calls the `SecureDataNodeStarter.init(...)` method while running as root. Once the `SecureDataNodeStarter` class has finished initializing, `jsvc` sets the effective UID to be the `hdfs` user, and then calls `SecureDataNodeStarter.start(...)`. `SecureDataNodeStarter` then calls the regular DataNode entry point, passing in a reference to the privileged resources it previously obtained.

MRv1 Only: The Linux TaskController

A setuid binary called `task-controller` is part of the `hadoop-0.20-mapreduce` package and is installed in either `/usr/lib/hadoop-0.20-mapreduce/sbin/Linux-amd64-64/task-controller` or `/usr/lib/hadoop-0.20-mapreduce/sbin/Linux-i386-32/task-controller`.

This `task-controller` program, which is used on MRv1 only, allows the TaskTracker to run tasks under the Unix account of the user who submitted the job in the first place. It is a setuid binary that must have a very specific set of permissions and ownership to function correctly. In particular, it must:

1. Be owned by root
2. Be owned by a group that contains only the user running the MapReduce daemons
3. Be setuid
4. Be group readable and executable

This corresponds to the ownership `root:mapred` and the permissions `4754`.

Here is the output of `ls` on a correctly-configured Task-controller:

```
-rwsr-xr-- 1 root mapred 30888 Mar 18 13:03 task-controller
```

The TaskTracker will check for this configuration on start up, and fail to start if the Task-controller is not configured correctly.

YARN Only: The Linux Container Executor

A setuid binary called `container-executor` is part of the `hadoop-yarn` package and is installed in `/usr/lib/hadoop-yarn/bin/container-executor`.

This `container-executor` program, which is used on YARN only and supported on GNU/Linux only, runs the containers as the user who submitted the application. It requires all user accounts to be created on the cluster hosts where the containers are launched. It uses a setuid executable that is included in the Hadoop distribution. The NodeManager uses this executable to launch and kill containers. The setuid executable switches to the user who has submitted the application and launches or kills the containers. For maximum security, this executor sets up restricted permissions and user/group ownership of local files and directories used by the containers such as the shared objects, jars, intermediate files, and log files. As a result, only the application owner and NodeManager can access any of the local files/directories including those localized as part of the distributed cache.

Parcel Deployments

In a parcel deployment the `container-executor` file is located inside the parcel at `/opt/cloudera/parcels/CDH/lib/hadoop-yarn/bin/container-executor`. For the `/usr/lib` mount point, setuid should not be a problem. However, the parcel could easily be located on a different mount point. If you are using a parcel, make sure the mount point for the parcel directory is without the nosuid option.

The `container-executor` program must have a very specific set of permissions and ownership to function correctly. In particular, it must:

1. Be owned by root
2. Be owned by a group that contains only the user running the YARN daemons
3. Be setuid
4. Be group readable and executable. This corresponds to the ownership `root:yarn` and the permissions `6050`.

```
---Sr-s--- 1 root yarn 91886 2012-04-01 19:54 container-executor
```

> **Important:** Configuration changes to the Linux container executor could result in local NodeManager directories (such as `usercache`) being left with incorrect permissions. To avoid this, when making changes using either Cloudera Manager or the command line, first manually remove the existing NodeManager local directories from all configured local directories (`yarn.nodemanager.local-dirs`), and let the NodeManager recreate the directory structure.

Troubleshooting

When you set up a secure cluster for the first time and debug problems with it, the `task-controller` or `container-executor` may encounter errors. These programs communicate these errors to the TaskTracker or NodeManager daemon via numeric error codes that appear in the TaskTracker or NodeManager logs respectively (`/var/log/hadoop-mapreduce` or `/var/log/hadoop-yarn`). The following sections list the possible numeric error codes with descriptions of what they mean:

- TaskController Error Codes (MRv1) on page 512
- ContainerExecutor Error Codes (YARN) on page 514

TaskController Error Codes (MRv1)

The following table applies to the task-controller in MRv1.

Numeric Code	Name	Description
1	INVALID_ARGUMENT_NUMBER	• Incorrect number of arguments provided for the given task-controller command • Failure to initialize the job localizer

Numeric Code	Name	Description
2	INVALID_USER_NAME	The user passed to the task-controller does not exist.
3	INVALID_COMMAND_PROVIDED	The task-controller does not recognize the command it was asked to execute.
4	SUPER_USER_NOT_ALLOWED_TO_RUN_TASKS	The user passed to the task-controller was the super user.
5	INVALID_TT_ROOT	The passed TaskTracker root does not match the configured TaskTracker root (`mapred.local.dir`), or does not exist.
6	SETUID_OPER_FAILED	Either could not read the local groups database, or could not set UID or GID
7	UNABLE_TO_EXECUTE_TASK_SCRIPT	The task-controller could not execute the task launcher script.
8	UNABLE_TO_KILL_TASK	The task-controller could not kill the task it was passed.
9	INVALID_TASK_PID	The PID passed to the task-controller was negative or 0.
10	ERROR_RESOLVING_FILE_PATH	The task-controller could not resolve the path of the task launcher script file.
11	RELATIVE_PATH_COMPONENTS_IN_FILE_PATH	The path to the task launcher script file contains relative components (for example, "..").
12	UNABLE_TO_STAT_FILE	The task-controller did not have permission to stat a file it needed to check the ownership of.
13	FILE_NOT_OWNED_BY_TASKTRACKER	A file which the task-controller must change the ownership of has the wrong the ownership.
14	PREPARE_ATTEMPT_DIRECTORIES_FAILED	The `mapred.local.dir` is not configured, could not be read by the task-controller, or could not have its ownership secured.
15	INITIALIZE_JOB_FAILED	The task-controller could not get, stat, or secure the job directory or job working working directory.
16	PREPARE_TASK_LOGS_FAILED	The task-controller could not find or could not change the ownership of the task log directory to the passed user.
17	INVALID_TT_LOG_DIR	The `hadoop.log.dir` is not configured.
18	OUT_OF_MEMORY	The task-controller could not determine the job directory path or the task launcher script path.
19	INITIALIZE_DISTCACHEFILE_FAILED	Could not get a unique value for, stat, or the local distributed cache directory.

Numeric Code	Name	Description
20	INITIALIZE_USER_FAILED	Could not get, stat, or secure the per-user task tracker directory.
21	UNABLE_TO_BUILD_PATH	The task-controller could not concatenate two paths, most likely because it ran out of memory.
22	INVALID_TASKCONTROLLER_PERMISSIONS	The task-controller binary does not have the correct permissions set. See Information about Other Hadoop Security Programs.
23	PREPARE_JOB_LOGS_FAILED	The task-controller could not find or could not change the ownership of the job log directory to the passed user.
24	INVALID_CONFIG_FILE	The taskcontroller.cfg file is missing, malformed, or has incorrect permissions.
255	Unknown Error	There are several causes for this error. Some common causes are: • There are user accounts on your cluster that have a user ID less than the value specified for the `min.user.id` property in the `taskcontroller.cfg` file. The default value is 1000 which is appropriate on Ubuntu systems, but may not be valid for your operating system. For information about setting `min.user.id` in the `taskcontroller.cfg` file, see this step. • Jobs do not run and the TaskTracker is unable to create a Hadoop logs directory. For more information, see (MRv1 Only) Jobs will not run and TaskTracker is unable to create a Hadoop logs directory. on page 498. • This error is often caused by previous errors; look earlier in the log file for possible causes.

ContainerExecutor Error Codes (YARN)

The codes in the table apply to the container-executor in YARN, but are used the LinuxContainerExecutor only.

Numeric Code	Name	Description
1	INVALID_ARGUMENT_NUMBER	• Incorrect number of arguments provided for the given task-controller command • Failure to initialize the container localizer
2	INVALID_USER_NAME	The user passed to the task-controller does not exist.
3	INVALID_COMMAND_PROVIDED	The container-executor does not recognize the command it was asked to run.

Numeric Code	Name	Description
5	INVALID_NM_ROOT	The passed NodeManager root does not match the configured NodeManager root (`yarn.nodemanager.local-dirs`), or does not exist.
6	SETUID_OPER_FAILED	Either could not read the local groups database, or could not set UID or GID
7	UNABLE_TO_EXECUTE_CONTAINER_SCRIPT	The container-executor could not run the container launcher script.
8	UNABLE_TO_SIGNAL_CONTAINER	The container-executor could not signal the container it was passed.
9	INVALID_CONTAINER_PID	The PID passed to the container-executor was negative or 0.
18	OUT_OF_MEMORY	The container-executor couldn't allocate enough memory while reading the container-executor.cfg file, or while getting the paths for the container launcher script or credentials files.
20	INITIALIZE_USER_FAILED	Couldn't get, stat, or secure the per-user NodeManager directory.
21	UNABLE_TO_BUILD_PATH	The container-executor couldn't concatenate two paths, most likely because it ran out of memory.
22	INVALID_CONTAINER_EXEC_PERMISSIONS	The container-executor binary does not have the correct permissions set. See Information about Other Hadoop Security Programs.
24	INVALID_CONFIG_FILE	The container-executor.cfg file is missing, malformed, or has incorrect permissions.
25	SETSID_OPER_FAILED	Could not set the session ID of the forked container.
26	WRITE_PIDFILE_FAILED	Failed to write the value of the PID of the launched container to the PID file of the container.
255	Unknown Error	This error has several possible causes. Some common causes are: • User accounts on your cluster have a user ID less than the value specified for the `min.user.id` property in the `container-executor.cfg` file. The default value is 1000 which is appropriate on Ubuntu systems, but may not be valid for your operating system. For information about setting `min.user.id` in the `container-executor.cfg` file, see this step. • This error is often caused by previous errors; look earlier in the log file for possible causes.