CDP Private Cloud Base 7.1.3

Encrypting Data at Rest in Cloudera Manager

Date published: 2020-07-10
Date modified: 2020-08-10

CLOUDERA

https://docs.cloudera.com/

Legal Notice

Contents

Encrypting Data at Rest

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 and Cloudera Navigator Key HSM..

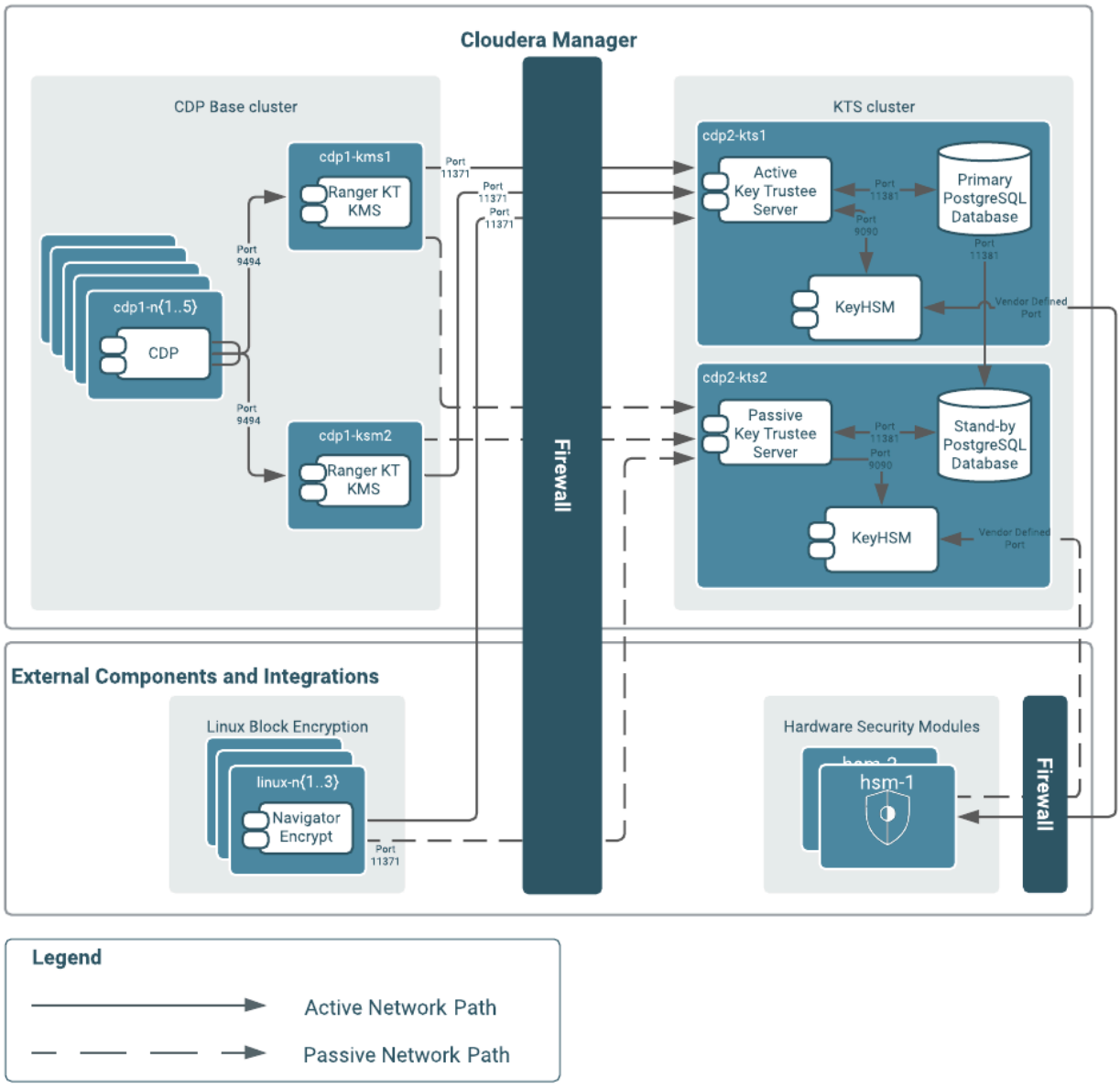
Before configuring encryption for data at rest, familiarize yourself with the requirements in "Data at Rest Encryption Requirements".

Related Information

Data at Rest Encryption Requirements

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 CDP 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 CDP services that require access to Key Trustee Server. This provides the following benefits:

- You can restart your CDP 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 "Resource Planning for Data at Rest Encryption".

Related Information

Resource Planning for Data at Rest Encryption

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 "Encrypting Data at Rest".

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\/u
random/' /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" 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 "Product Compatibility Matrix for Cloudera Navigator Encryption".

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.

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 "Cloudera Enterprise Upgrade Guide"> 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 "Registering Cloudera Navigator Encrypt with Key Trustee Server>Registration Options" />.

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 "Product Compatibility Matrix for Cloudera Navigator Encryption".

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 "Product Compatibility Matrix for Cloudera Navigator Encryption".
- 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

The supported Linux distribution, which is listed in the "Product Compatibility Matrix for Cloudera Navigator Encryption".

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

- For supported Linux distributions, see "Product Compatibility Matrix for Cloudera Navigator Encryption".

Supported command-line interpreters:

- sh (Bourne)
- bash (Bash)
- dash (Debian)

Note: Navigator Encrypt does not support installation or use in chroot environments.

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.

Related Information

Encrypting Data at Rest
Resource Planning for Data at Rest Encryption
Product Compatibility Matrix for Cloudera Navigator Encryption
Registering Cloudera Navigator Encrypt with Key Trustee Server
AES-NI
JCE for Java SE 7
JCE for Java SE 8

Resource Planning for Data at Rest Encryption

For production environments, you must configure high availability for:

- Key Trustee Server
- Key Trustee 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" 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" 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" for more information.

For information about enabling Key Trustee Server high availability, refer to "Cloudera Navigator Key Trustee Server High Availability".

For information about enabling Key Trustee KMS high availability, refer to "Enabling Key Trustee 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.

Related Information

Data at Rest Encryption Requirements
Data at Rest Encryption Reference Architecture
RHEL 6 Security Guide
RHEL 7 Security Guide
AES-NI
Cloudera Navigator Key Trustee Server High Availability
Enabling Key Trustee KMS High Availability

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.

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 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.

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.

Related Information

Transparent Encryption in HDFS
AES-NI Standard Instructions
Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7 Download
Optimizing Performance for HDFS Transparent Encryption

Key Concepts and Architecture

HDFS must be integrated with an external enterprise-level keystore. The Hadoop Key Management server serves as a proxy between HDFS clients and the keystore. The keystore can be either the Cloudera Navigator Key Trustee Server or a support Hardware Security Module.

HDFS transparent encryption involves the creation of an encryption zone, which is a directory in HDFS whose contents will be automatically encrypted on write and decrypted on read. Each encryption zone is associated with a key (EZ Key) specified by the key administrator when the zone is created. The EZ keys are stored on the external keystore.

Keystores and the Hadoop Key Management Server

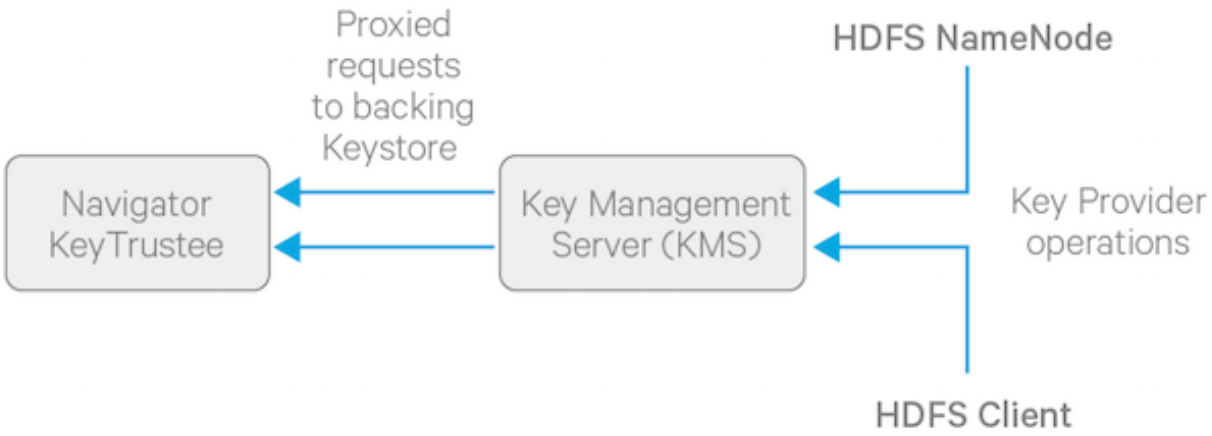
Integrating HDFS with an external, enterprise-level keystore is the first step to deploying transparent encryption. This is because separation of duties between a key administrator and an HDFS administrator is a very important aspect of this feature.

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.

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.



Data Encryption Components and Solutions

Cloudera supports four encryption components which may be combined as unique solutions. When selecting a Key Management System (KMS), you must decide which components meet the key management and encryption requirements for your enterprise.

Cloudera Encryption components

Descriptions of Cloudera components for encrypting data at rest follow:

Ranger Key Management System (KMS)

Ranger extends the native Hadoop KMS functionality by allowing you to store keys in a secure database or you can use the secure key store like Key Trustee Server. Cryptographic key management service supporting HDFS TDE. Not a general purpose Key Management System, as opposed to Hadoop KMS which stores keys in file based Java Keystore, can be accessed only through KeyProvider API.

Key Trustee Server (KTS)

Key Manager that stores and manages cryptographic keys and other security artifacts

Key HSM

Allows Ranger Key Trustee Server to seamlessly integrate with the following hardware security modules (HSM)

- Luna 6 & 7
- CipherTrust
- GCP Cloud HSM
- Azure Key Vault

Navigator Encrypt

Transparently encrypts and secures data at rest without requiring changes to your applications

Cloudera Encryption solutions

You can deploy encryption components as any of the following solutions for encrypting data at rest:

Ranger KMS Only

- Consists of ONLY Ranger KMS with a backend database that provides key storage
- Ranger KMS provides enterprise-grade key management

Ranger KMS + HSM

- Consists of Ranger KMS with database + integration with a backend hardware security module (HSM)
- Ranger KMS provides enterprise-grade key management
- HSM provides encryption zone key protection
- HSM stores only the encryption master key

Ranger KMS + Key Trustee Server (KTS)

- Ranger KMS provides enterprise-grade key management
- KTS provides the key store that stores and manages cryptographic keys and other security artifacts

Ranger KMS + KTS + Key HSM

Allows Cloudera Key Trustee Server to seamlessly integrate with a HSM in addition to above items

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 with the -skipcrccheck -update flags can be used to add data to a zone. (These flags are required because encryption zones are being used.) Every file and subdirectory copied to an encryption zone will be encrypted.

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. 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.

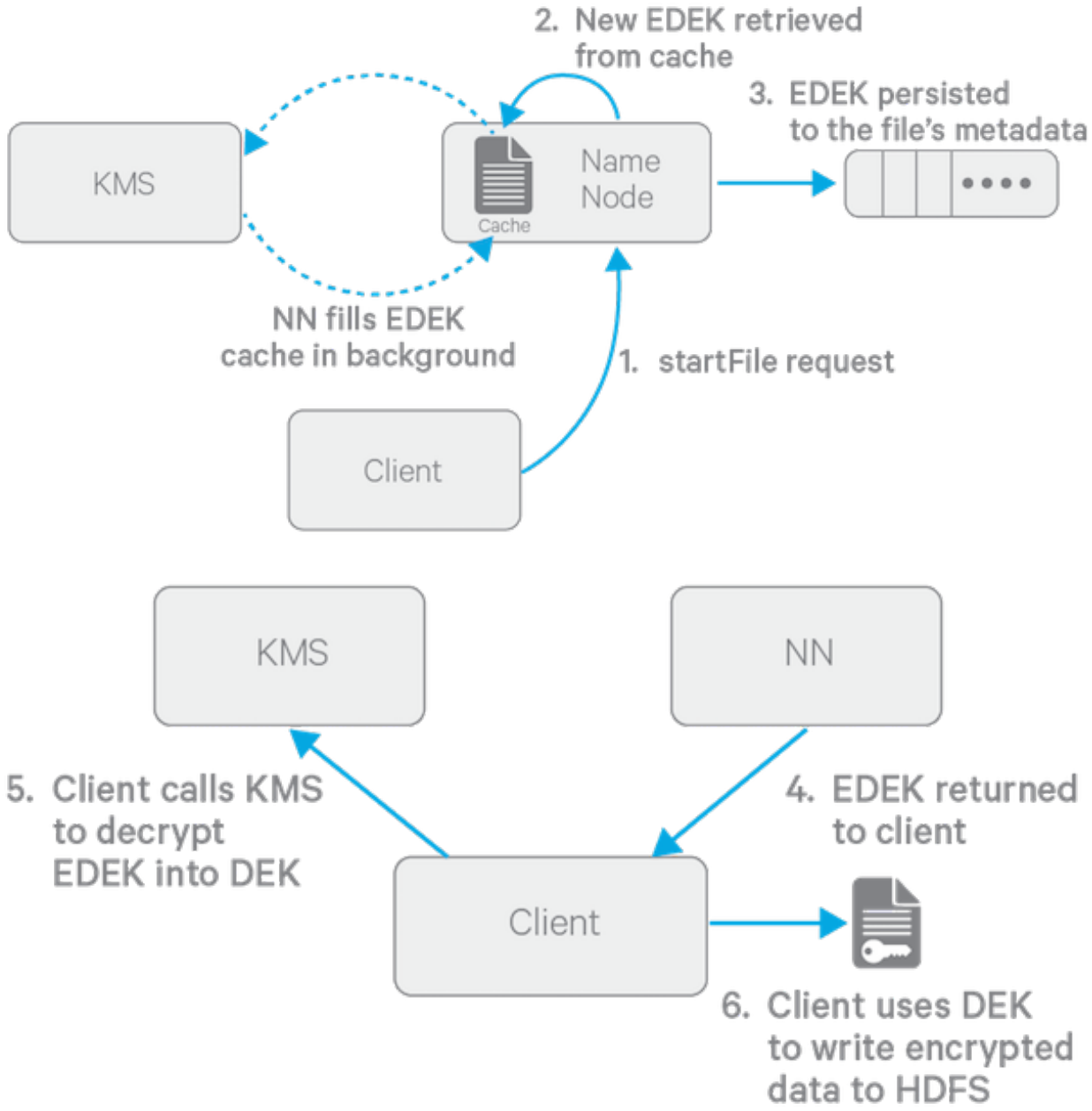
Related Information

Managing Encryption Keys and Zones

Extended Attributes in HDFS

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

CDP 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 CDP supports.

Warning: To ensure that HDFS encryption functions as expected, the steps described in this section are mandatory for production use.

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/lib
   crypto.so
   ```

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
- 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. More specifically, the Java KeyStore KMS does not provide:

- Scalability, so you are limited to only one KMS, which can result in bottlenecks
- High Availability (HA)
- Recoverability, so if you lose the node where the Java KeyStore is stored, then you can lose access to all the encrypted data

Ultimately, the Java KeyStore does not satisfy the stringent security requirements of most organizations for handling master encryption keys.

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.

Related Information

Encrypting Data at Rest
Data at Rest Encryption Requirements
Data at Rest Encryption Reference Architecture
Resource Planning for Data at Rest Encryption

Enabling HDFS Encryption Using Navigator Key Trustee Server

Enabling HDFS encryption using Key Trustee Server as the key store involves multiple components.

Before you begin

Before continuing, make sure the Cloudera Manager server host has access to the internal repository hosting the Key Trustee Server software.

About this task

Selecting Cloudera Navigator Key Trustee Server as the root of trust to view the various steps.

Procedure

1. Enable Kerberos.
   Minimum Required Role: Cluster Administrator (also provided by Full Administrator)
2. Enable TLS/SSL.
   Minimum Required Role: Cluster Administrator (also provided by Full Administrator)

3. Add a dedicated cluster for the Key Trustee Server.

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

   If you have not already done so, you must create an internal repository to install Cloudera Navigator before you can set up and use Navigator Key Trustee Server.

   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. To complete this step:

   a) Click Add a dedicated cluster for the Key Trustee Server.
   b) 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.
   c) 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.
   d) 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 step *4. Install Key Trustee Server binary using packages or parcels.*
   e) After the KEYTRUSTEE_SERVER parcel is successfully downloaded, distributed, unpacked, and activated, click Continue.
   f) 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)

   Important: If you selected None in *3. Add a dedicated cluster for the Key Trustee Server.*, 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*.

   If you have not already done so, you must create an internal repository to install Cloudera Navigator before you can set up and use Navigator Key Trustee Server.

   This step is completed automatically during step *3. Add a dedicated cluster for the Key Trustee Server.* 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:

   a) Click Install Key Trustee Server binary using packages or parcels.
   b) 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.
   c) 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)

   If you have not already done so, you must create an internal repository to install Cloudera Navigator before you can set up and use Navigator Key Trustee Server.

   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*.

   To complete this step for parcel-based installations:

   a) Click Install Parcel for Key Trustee KMS.
   b) 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.
   c) 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:

   a) Click Add a Key Trustee Server Service.
   b) Click Continue.
   c) 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 and click Continue.
   d) The Entropy Considerations page provides commands to install the rng-tools package to increase available entropy for cryptographic operations.

      After completing these commands, click Continue.
   e) 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.

f) 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).

Click Continue to view and modify the default certificate settings.

g) 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 (if any intermediate certificates exist, then they are required here) 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 (if any intermediate certificates exist, then they are required here) 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.

h) 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.

i) 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 ten backups are retained in /var/lib/keytrustee in cleartext.

7. 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:

a) Click Restart stale services and redeploy client configuration.
b) Click Restart Stale Services.
c) Ensure that Re-deploy client configuration is checked, and click Restart Now.
d) After all commands have completed, click Finish.

8. 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.

Related Information

Resource Planning for Data at Rest Encryption
Data at Rest Encryption Reference Architecture
Data at Rest Encryption Requirements
Configuring TLS/SSL for the KMS

Enabling HDFS Encryption Using a 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.

About this task

After selecting A file-based password-protected Java KeyStore as the root of trust, the following steps are displayed.

Procedure

1. Enable Kerberos.

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

2. Enable TLS/SSL.

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

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:

   a) Click Add a Java KeyStore KMS Service.
   b) Select a cluster host for the Java KeyStore KMS service. Click Continue.
   c) 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.

      Click Continue.
   d) 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.
4. Click Continue to automatically configure the HDFS service to depend on the Java KeyStore KMS service.
5. Click Finish to complete this step and return to the main page of the wizard.
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:

   a) Click Restart stale services and redeploy client configuration.
   b) Click Restart Stale Services.
   c) Ensure that Re-deploy client configuration is checked, and click Restart Now.
   d) After all commands have completed, click Finish.

7. Validate Data Encryption.

   Minimum Required Role: Key Administrator or Cluster 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.

Related Information

Configuring TLS/SSL for the KMS

Managing Encryption Keys and Zones

Interacting with the KMS and creating encryption zones requires the use of two CLI commands: hadoop key and hdfs crypto. Before getting started with creating encryption keys and setting up encryption zones, make sure that your KMS ACLs have been set up according to best practices.

Validating Hadoop Key Operations

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
```

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.

Creating Encryption Zones

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.

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.crypto.key.JavaKeyStoreProvider) does not support uppercase key names.

- Create an encryption key for your zone as keyadmin for the user/group (regardless of the application that will be using the encryption zone):

  ```
  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. Also, do not delete the KMS service, as your encrypted HDFS data will be inaccessible. To prevent data loss, first copy the encrypted HDFS data to a non-encrypted zone using the distcp command.

Related Information

Configuring CDP Services for HDFS Encryption

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: You can delete files or directories that are part of an HDFS encryption zone.

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, the virtual path prefix /.reserved/raw/ has been introduced, 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 data from encrypted locations

By default, distcp compares checksums provided by the filesystem to verify that data was successfully copied to the destination. When copying from an encrypted location, the file system checksums will not match because the underlying block data is different. This is true whether or not the destination location is encrypted or unencrypted.

In this case, you can specify the -skipcrccheck and -update flags to avoid verifying checksums. When you use -ski pcrccheck, 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

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.

Rolling Encryption Keys

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.

Before you begin

Before attempting to roll an encryption key (also known as an encryption zone key, or EZ key),you must be familiar with the concepts associated with Navigator Data Encryption and the HDFS Transparent Encryption.

About this task

You may want to roll the encryption key periodically, as part of your security policy or when an external security compromise is detected.

Procedure

1. 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. 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}. crypto
   ProtocolVersion:
   CryptoProtocolVersion{description=’Encryption zones’, version=2, unknow
   nValue=null}, edek: 373c0c2e919c27e58c1c343f54233cbd,
   iv: d129c913c8a34cde6371ec95edfb7337, keyName: key1, ezKeyVersionName:
    7mbvopZ0Weuvs0XtTkpGw3G92KuWc4e4xcTXl0bXCpF}
   ```

3. Log off as HDFS Superuser and 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.L
   oadBalancingKMSClientProvider@5ea434c8
   for keyName: key1
   key1 has been successfully rolled.
   org.apache.hadoop.crypto.key.kms.LoadBalancingKMSClientProvider@5ea434c8
    has been updated.
   ```

4. 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}. cryptoP
   rotocolVersion:
   CryptoProtocolVersion{description=’Encryption zones’, version=2, unknown
   Value=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_HOS
TNAME>: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)

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. 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.

Related Information

Rolling Encryption Keys

Benefits and Capabilities

In addition to minimizing security risks, re-encrypting the EDEK offers other 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

There are certain considerations that you must be aware of as you re-encrypt an EDEK.

- 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.

- 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

There are few limitations associated with the re-encryption of EDEKs.

Caution: You cannot perform any rename operations within the encryption zone during re-encryption. If you attempt to perform a rename operation during EDEK re-encryption, you will receive an IOException error.

EDEK re-encryption does not 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.

Procedure

1. Navigate to the cluster in which you will be rolling keys and re-encrypting the EDEK.
2. Log in as HDFS Superuser.
3. 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.L
oadBalancingKMSClientProvider@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. 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}. crypto
ProtocolVersion:
CryptoProtocolVersion{description='Encryption zones', version=2, unknow
nValue=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
```

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.	ZMHfRoGKeXXgf0QzCX8q16NczIw2sq0rWRTOHS3YjCz
Submission Time	The time at which the re-encryption 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

Column Name	Description	Sample Data
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
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. 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}. crypto
ProtocolVersion:
CryptoProtocolVersion{description='Encryption zones', version=2, unknow
nValue=null}, edek: 373c0c2e919c27e58c1c343f54233cbd,
iv: d129c913c8a34cde6371ec95edfb7337, keyName: key1, ezKeyVersionName:
 ZMHfRoGKeXXgf0QzCX8q16NczIw2sq0rWRTOHS3YjCz }
```

Managing Re-encryption Operations

There are various facets of the EDEK re-encryption process such as cancelling re-encryption, rolling keys during a re-encryption operation, and throttling re-encryption operations.

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 Jetty server bundled with the Hadoop distribution.

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. Cloudera strongly recommends not using Java Keystore KMS in production environments. Java KeyStore KMS does not support high availability.
- 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. 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.

Related Information

Securing the Key Management System (KMS)

Configuring the KMS Using Cloudera Manager

If you are using Cloudera Manager, you can view and edit the KMS configuration by navigating to the specific pages, depending on the KMS implementation you are using.

- Key Trustee KMS service Configuration

- Java KeyStore KMS service Configuration

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 serviceConfigurationAdvancedKey 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>
```

Configuring the Audit Log Aggregation Interval

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 serviceConfigurationAdvancedKey Management Server Proxy Advanced Configuration Snippet (Safety Valve) for kms-site.xml:

```
<property>
    <name>hadoop.kms.aggregation.delay.ms</name>
    <value>10000</value>
</property>
```

Securing the Key Management System (KMS)

Cloudera provides the following Key Mangement System (KMS) implementations: Ranger KMS with database, Ranger KMS with HSM, Ranger KMS with Key Trustee Server, and Ranger KMS with Key Trustee Server and Key HSM. You can secure Ranger KMS using Kerberos, TLS/SSL communication, and access control lists (ACLs) for operations on encryption keys.

Cloudera Manager supports wizard-driven instructions for installing both Ranger KMS with a database and Ranger KMS with KTS.

Enabling Kerberos Authentication for the KMS

You can use Cloudera Manager to enable Kerberos authentication for the KMS.

About this task
Minimum Required Role: Full Administrator

Procedure
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.

Configuring TLS/SSL for the KMS
You must configure specific TLS/SSL properties associated with the KMS.

About this task
Minimum Required Role: Configurator (also provided by Cluster Administrator, Full Administrator)

Procedure
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.

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.
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.

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.

Before you begin

This procedure assumes that the Java KeyStore (JKS) is on the same host as the new Key Trustee KMS service.

Procedure

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.
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/jetty-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/jetty-deployment/webapps/kms/WEB-INF/classes/*tmp/password.txt* to kms:kms.

```
sudo chown kms:kms /var/lib/keytrustee-kms/jetty-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.n
ame=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.n
ame=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.

Related Information

Role Instances

Migrating Ranger Key Management Server Role Instances to a New Host

You can move the Ranger Admin, Ranger KMS db and Ranger KMS KTS role instances for an existing Ranger KMS service from one host to another, using Cloudera Manager.

Note: This procedure applies only to the Ranger Key Management Server role instances. Do not attempt to move the Key Trustee Server.

In some cases–for example, after upgrading your servers–you may want to migrate a Ranger KMS Server role instance to a new host. This procedure describes how to move a Ranger KMS role instance from an existing cluster host to another cluster host.

Migrate the Ranger Admin role instance to a new host

To migrate the Ranger KMS role instances to a new host, first migrate the Ranger Admin role instance.

Procedure

1. Add a new Ranger Admin role instance on another node.

   Note: If you enabled manual SSL on this cluster, you must update the SSL configs when adding a new role.

2. Start the new Ranger Admin role instance.
3. Stop the initial Ranger Admin instance.
4. Delete the initial Ranger Admin instance.
5. Restart the cluster.

   Restarting the cluster removes the "stale" changes.

Migrate the Ranger KMS db role instance to a new host

After migrating the Ranger Admin role instance to a new host, migrate the Ranger KMS db role instance.

About this task

Only if Ranger KMS has a backend database for key storage, should you migrate the Ranger KMS db role instance.

Procedure

1. Add a new Ranger KMS db role instance on another node.

   Note: If you enabled manual SSL on this cluster, you must update the SSL configs when adding a new role.

2. Start the new Ranger KMS db role instance.
3. Stop the initial Ranger KMS db instance.
4. Delete the initial Ranger KMS db instance.
5. Restart the cluster.
6. Login to Ranger Admin UI using keyadmin credentials.
7. Update the cm_kms service to use the kms url that refers to the new hostname.

Migrate the Ranger KMS KTS role instance to a new host

After migrating the Ranger Admin, Ramger KMS db role instances to a new host, migrate the Ranger KMS KTS role instance.

About this task

Only if Ranger KMS is backed by Key Trustee Server for key storage, should you migrate the Ranger KMS KTS role instance.

Procedure

1. Add a new Ranger KMS KTS role instance on another node.

   Note: If you enabled manual SSL on this cluster, you must update the SSL configs when adding a new role.

2. Start the new Ranger KMS KTS role instance.
3. Stop the initial Ranger KTS service.
4. Delete the older Ranger KTS instance.
5. Restart the cluster.
6. Login to Ranger Admin UI using keyadmin credentials.
7. Update the cm_kms service to use the kms url that refers to the new hostname.
8. Copy or rsync conf and gpg files such as keytrustee.conf, pubring.gpp and secring.gpg present at /var/lib/kms-keytrustee/keytrustee/.keytrustee/ from older host to new host. For example:

```
[root@mm-ktslog-1 ~]# ll /var/lib/kms-keytrustee/keytrustee/.keytrustee/
total 20
-rw------- 1 kms kms  715 Oct  7 10:59 keytrustee.conf
-rw------- 1 kms kms 5026 Oct  7 10:59 pubring.gpg
-rw------- 1 kms kms 4885 Oct  7 10:59 secring.gpg
```

9. Restart the Ranger KMS KTS service.

Migrating ACLs from Key Trustee KMS to Ranger KMS

Perform the procedures in this section to migrate ACLs from Key Trustee Key Management Server (KMS) to Ranger KMS.

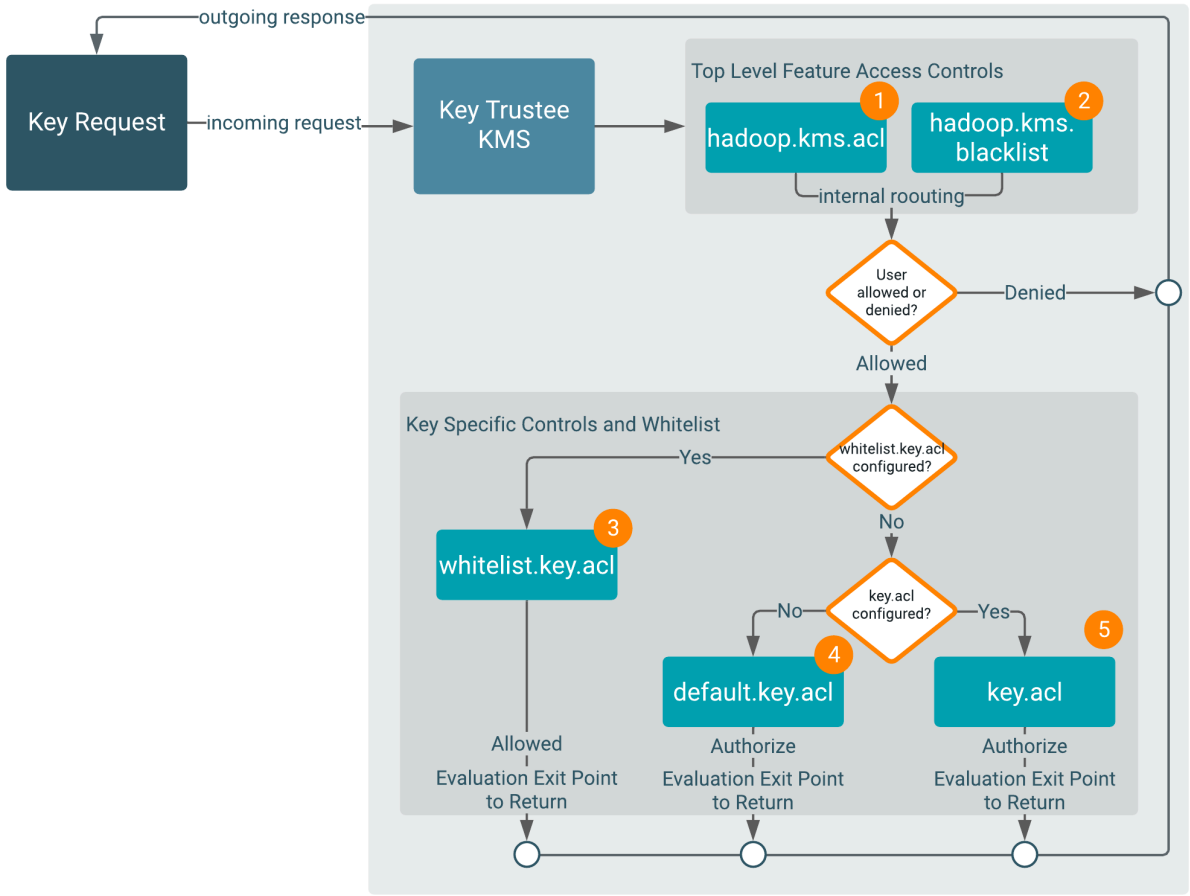
Key Trustee ACL evaluation

Before going into the details of how Key Trustee ACLs are evaluated, it is critical that you understand the key rules that the Key Trustee Key Management Server uses in performing this evaluation.

KMS ACL Flow Rules:

- The whitelist class bypasses key.acl and default.key.acl controls.
- The key.acl definitions override all default definitions.

Encryption key access is evaluated as follows:



1 and 2

The KMS evaluates the hadoop.kms.acl.<OPERATION> and hadoop.kms.blacklist.<OPERATION> classes to determine whether or not access to a specific KMS feature or function is authorized.

In other words, a user must be allowed by hadoop.kms.acl.<OPERATION>, and not be disallowed by hadoop.kms.blacklist.<OPERATION>.

If a user is denied access to a KMS-wide operation, then the flow halts and returns the result Denied.

If a user is allowed access to a KMS-wide operation, then the evaluation flow proceeds.

3

The KMS evaluates the whitelist.key.acl class.

The KMS ACL workflow evaluates the whitelist.key.acl.<OPERATION>, and if the user is allowed access, then it is granted (Allowed) . If not, then the flow continues with the evaluation.

4 and 5

The KMS evaluates the default.key.acl.<OPERATION> and key.acl.<OPERATION> classes.

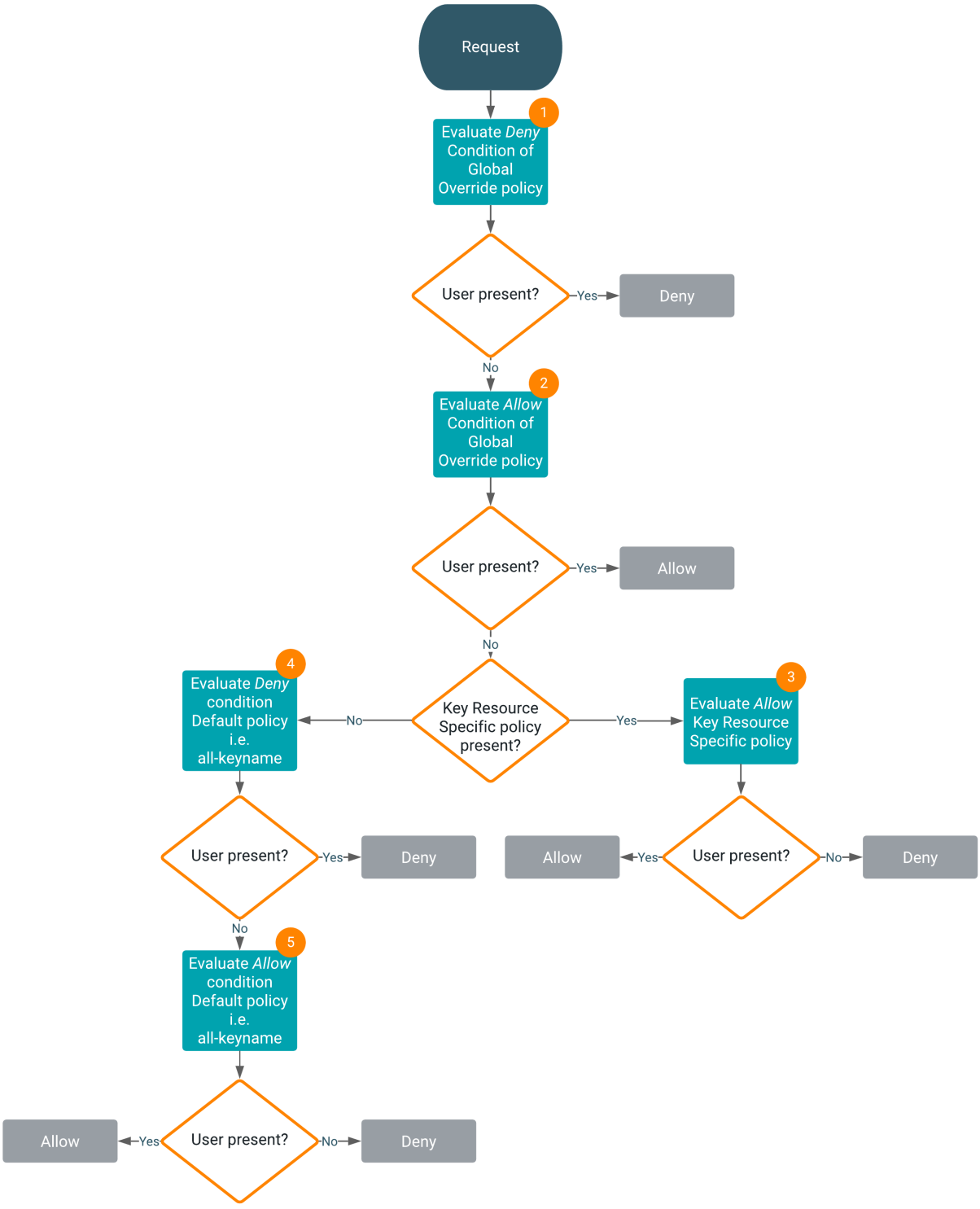
The KMS evaluates whether or not there is a key.acl.KEY.<OPERATION> class that matches the action the user is attempting to perform. If there is, it then evaluates that value to determine whether or not the user can perform the requested operation.

Note: Before evaulating the default.key.acl.<OPERATION> and key.acl.<OPERATION> classes, the flow logic determines which classes exist. Only one of these can exist and be used at any time (for example, key.acl.prodkey.READ overrides default.key.acl.READ for prodkey, so the flow logic is configured with it's own READ ACLs)

Depending on the result of the Key Trustee ACL evaluation, controls are applied to the key and results (Allowed or Denied).

Access evaluation with Ranger KMS policies

Access is evaluated with Ranger KMS policies as follows:



1

After the request is received, the Deny condition of the Global Override policy is evaluated. If the user is present, the flow halts and returns the result Deny. If the user is not present, the evaluation flow proceeds.

2

Now, the Allow condition of the Global Override policy is evaluated. If the user is present, the flow halts and returns the result Allow. If the user is not present, the evaluation flow proceeds.

3

If the Key Resource Specific policy is present, the Allow condition of the Key Resource Specific policy is evaluated. If the user is not present, the flow halts and returns the result Deny. If the user is present, the flow is complete and returns the result Allow.

4

If the Key Resource Specific policy is not present, the Deny condition of the Default policy, all-keyname, is evaluated. If the user is present, the flow halts and returns the result Deny. If the user is not present, the evaluation flow proceeds.

5

Now, the Allow condition of the Default policy, all-keyname, is evaluated. If the user is not present, the flow halts and returns the result Deny. If the user is present, the flow is complete and returns the result Allow.

Key Trustee KMS operations not supported by Ranger KMS

The Key Trustee KMS operations mentioned here are not supported by Ranger KMS.

- hadoop.kms.acl.<OPERATION>

  The ACLs mentioned below are ignored by Ranger KMS because these ACLs are not migrated to the Ranger KMS policy.

```
hadoop.kms.acl.CREATE
hadoop.kms.acl.DELETE
hadoop.kms.acl.ROLLOVER
hadoop.kms.acl.GET
hadoop.kms.acl.GET_KEYS
hadoop.kms.acl.GET_METADATA
hadoop.kms.acl.SET_KEY_MATERIAL
hadoop.kms.acl.GENERATE_EEK
hadoop.kms.acl.DECRYPT_EEK
```

- keytrustee.kms.acl.<OPERATION>

  The ACLs mentioned below are Key Trustee-specific ACLs. These ACLs are ignored by Ranger KMS because they are not migrated to the Ranger KMS policy. Also, these ACLs are not supported by Hadoop KMS.

```
keytrustee.kms.acl.UNDELETE
keytrustee.kms.acl.PURGE
```

ACLs supported by Ranger KMS and Ranger KMS Mapping

The ACLs mentioned here are supported by Ranger KMS and Ranger KMS mapping.

- whitelist.key.acl.<operation> and hadoop.kms.blacklist.<Operation>

  In this case, you create a Global Override policy under the service cm_kms.

  Service : cm_kms

Policy	Key-resource	Priority	Key Trustee ACL	Ranger Policy Condition	Ranger Policy Permission
Global Override Policy	*	Override	whitelist.key.acl.MANAGEMENT	ALLOW	CREATE, DELETE, ROLLOVER
			whitelist.key.acl.GENERATE_EEK	ALLOW	GENERATE_EEK
			whitelist.key.acl.DECRYPT_EEK	ALLOW	DECRYPT_EEK

Policy	Key-resource	Priority	Key Trustee ACL	Ranger Policy Condition	Ranger Policy Permission
			whitelist.key.acl.READ	ALLOW	GET, GET KEYS, GET METADATA
			hadoop.kms.blacklist.CREATE	DENY	CREATE
			hadoop.kms.blacklist.DELETE	DENY	DELETE
			hadoop.kms.blacklist.ROLLOVER	DENY	ROLLOVER
			hadoop.kms.blacklist.GET	DENY	GET
			hadoop.kms.blacklist.GET_KEYS	DENY	GET KEYS
			hadoop.kms.blacklist.GET_METADATA	DENY	GET METADATA
			hadoop.kms.blacklist.SET_KEY_MATERIAL	DENY	SET KEY MATERIAL
			hadoop.kms.blacklist.GENERATE_EEK	DENY	GENERATE_EEK
			hadoop.kms.blacklist.DECRYPT_EEK	DENY	DECRYPT_EEK

- default.key.acl.<operation>

  Service : cm_kms

Policy	Key-resource	Priority	Key Trustee ACL	Ranger Policy Condition	Ranger Policy Permission
Default Policy all-keyname	*	Normal	default.key.acl.MANAGEMENT	ALLOW	CREATE, DELETE, ROLLOVER
			default.key.acl.GENERATE_EEK	ALLOW	GENERATE_EEK
			default.key.acl.DECRYPT_EEK	ALLOW	DECRYPT_EEK
			default.key.acl.READ	ALLOW	GET, GET KEYS, GET METADATA

- key.acl.<key-name>.<OPERATION> Key Specific ACL

  In this case, you create a Key Resource Specific policy under the service cm_kms.

  Service : cm_kms

Policy	Key-resource	Priority	Key Trustee ACL	Ranger Policy Condition	Ranger Policy Permission
Key Resource Specific policy <keyname>	<keyname>	Normal	key.acl.<key-name>.MANAGEMENT	ALLOW	CREATE, DELETE, ROLLOVER
			key.acl.<key-name>.GENERATE_EEK	ALLOW	GENERATE_EEK
			key.acl.<key-name>.DECRYPT_EEK	ALLOW	DECRYPT_EEK
			key.acl.<key-name>.READ	ALLOW	GET, GET KEYS, GET METADATA

Policy	Key-resource	Priority	Key Trustee ACL	Ranger Policy Condition	Ranger Policy Permission
			key.acl.<key-name>.ALL	ALLOW	SELECT ALL

Note: In Key Resource Specific policies, DENY ALL OTHER ACCESS flags are set to true.

Configuring CDP Services for HDFS Encryption

There are recommendations that you must consider for setting up HDFS Transparent Encryption with various CDP services.

Important: Encrypting /tmp using HDFS encryption is not supported.

Transparent Encryption Recommendations for HBase

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 ACL, 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>
```

Transparent Encryption Recommendations for 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 IN PATH 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 ACL, 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.

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

You must consider various factors when working with Hive tables after enabling HDFS transparent encryption for Hive.

- 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 * FR
OM <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 ACL 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, 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>
```

Transparent Encryption Recommendations for Hue

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>
```

Transparent Encryption Recommendations for Impala

There are various recommendations to consider when configuring HDFS Transparent Encryption for Impala.

Recommendations

- If HDFS encryption is enabled, configure Impala to encrypt data spilled to local disk.
- 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 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 serviceConfigurationCategorySecurity).

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.

Related Information

KMS ACL Configuration for Hive

Transparent Encryption Recommendations for MapReduce and YARN

MapReduce v1 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.

Recommendations for MapReduce v2 (YARN)

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.

Transparent Encryption Recommendations for Search

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 ACL, 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>
```

Transparent Encryption Recommendations for Spark

There are various recommendations to consider when configuring HDFS Transparent Encryption for 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 ACL, 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>
```

Transparent Encryption Recommendations for Sqoop

There are various recommendations to consider when configuring HDFS Transparent Encryption for 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 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.

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”, 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.
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/jetty-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/jetty-deployment/webapps/kms/WEB-INF/classes*/tmp/password.txt* to kms:kms:

```
sudo chown kms:kms /var/lib/keytrustee-kms/jetty-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/tr
ustee/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" "htt
ps://kms01.example.com:16000/kms/v1/trustee/key/migrate?user.n
ame=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 CDP Services for HDFS Encryption

This page contains recommendations for setting up HDFS Transparent Encryption with various CDP 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 (“Add the HttpFS role”) 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 (“Configuring KMS Access Control Lists (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 ACL (“Configuring KMS Access Control Lists (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 IN PATH 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 (“Configuring KMS Access Control Lists (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 52.

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 * FR
      OM <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 (“Configuring KMS Access Control Lists (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 Security Configuration”, 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 (“Configuring KMS Access Control Lists (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 (“Configuring KMS Access Control Lists (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 serviceConfigurationCategorySecurity).

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 53.

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 (“Configuring KMS Access Control Lists (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 (“Configuring KMS Access Control Lists (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 (“Configuring KMS Access Control Lists (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 (“Configuring KMS Access Control Lists (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 (“Configuring KMS Access Control Lists (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 51.
- 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.

Using the Ranger Key Management Service

Ranger KMS can be accessed by logging into the Ranger web UI as the KMS administrator.

Role Separation

Ranger uses separate admin users for Ranger and Ranger KMS.

- The Ranger admin user manages Ranger access policies.
- The Ranger KMS admin user (keyadmin by default) manages access policies and keys for Ranger KMS, and has access to a different set of UI features than the Ranger admin user.

Using separate administrator accounts for Ranger and Ranger KMS separates encryption work (encryption keys and policies) from cluster management and access policy management.

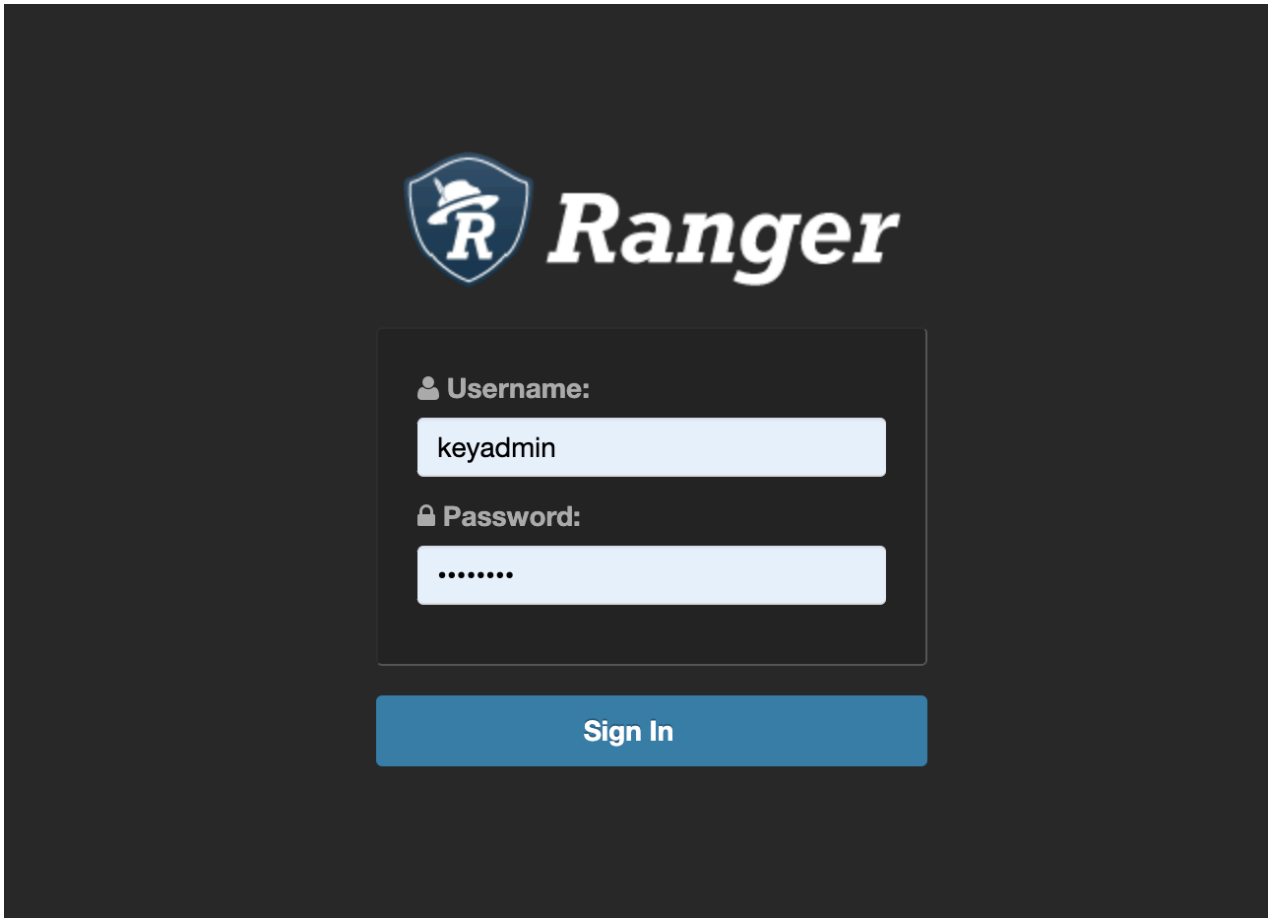
Note:

For more information about creating, deleting, listing, and rolling over existing keys using Ranger REST APIs, see https://ranger.apache.org/apidocs/resource_XKeyREST.html.

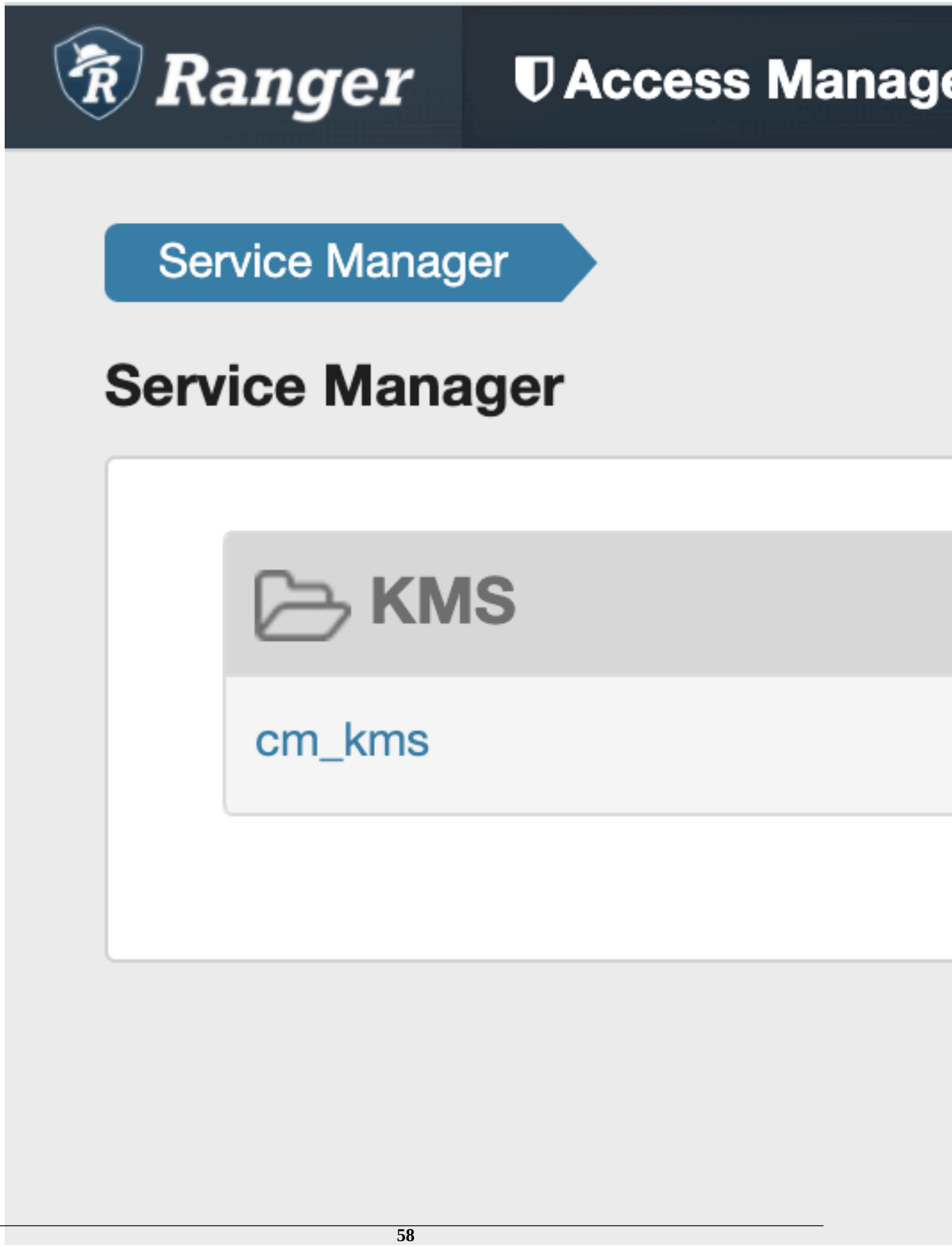
Accessing the Ranger KMS Web UI

How to access the Ranger Key Management Service web UI.

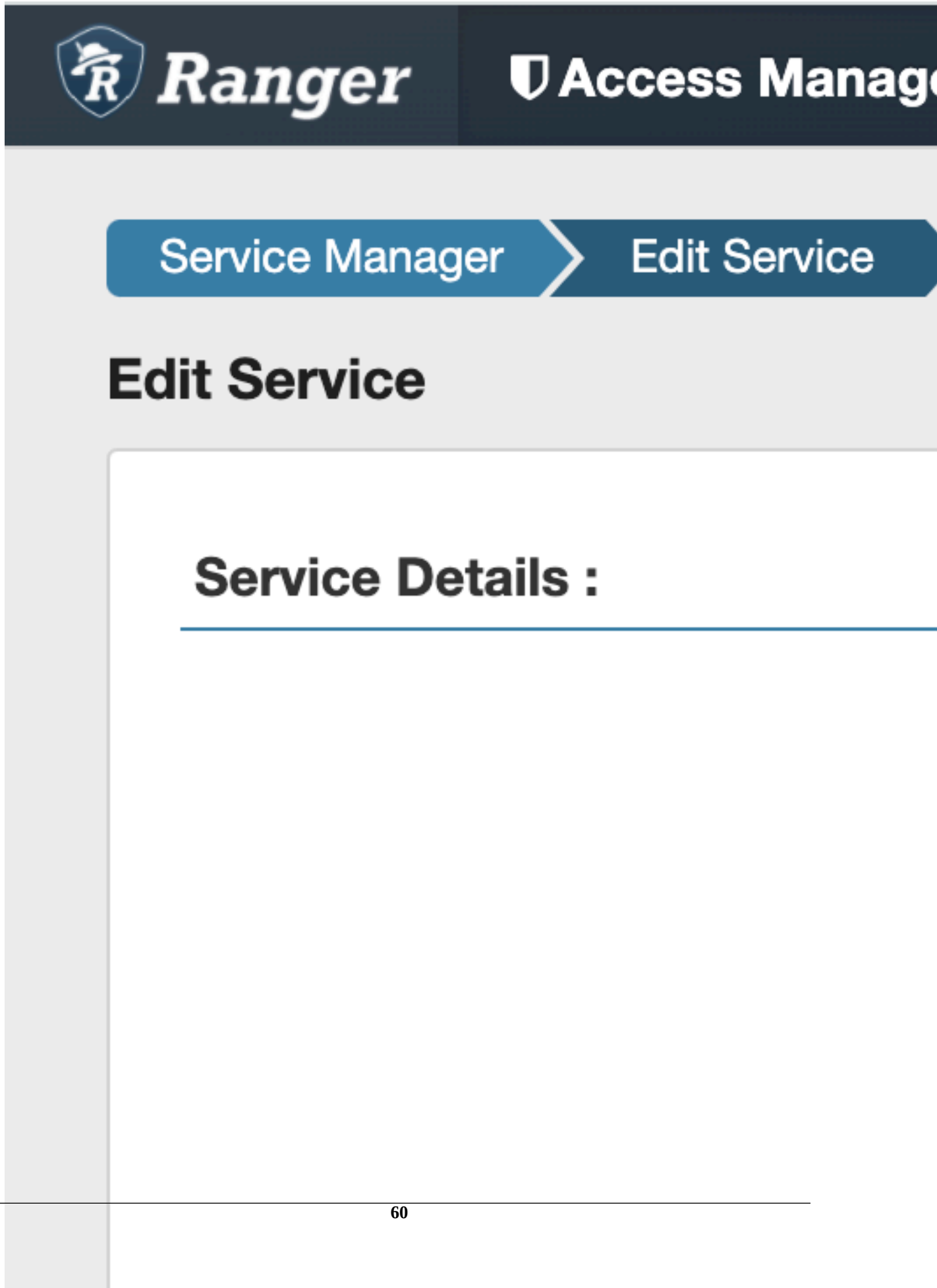
To access Ranger KMS, click the Ranger Admin web UI link, enter your Ranger KMS admin user name and password, then click Sign In.



After logging in, the Service Manager page appears.


Ranger
Access Manage
Service Manager
Service Manager
KMS
cm_kms

To edit Ranger KMS repository properties, click the Edit icon for the service and update the settings on the Edit Service page.

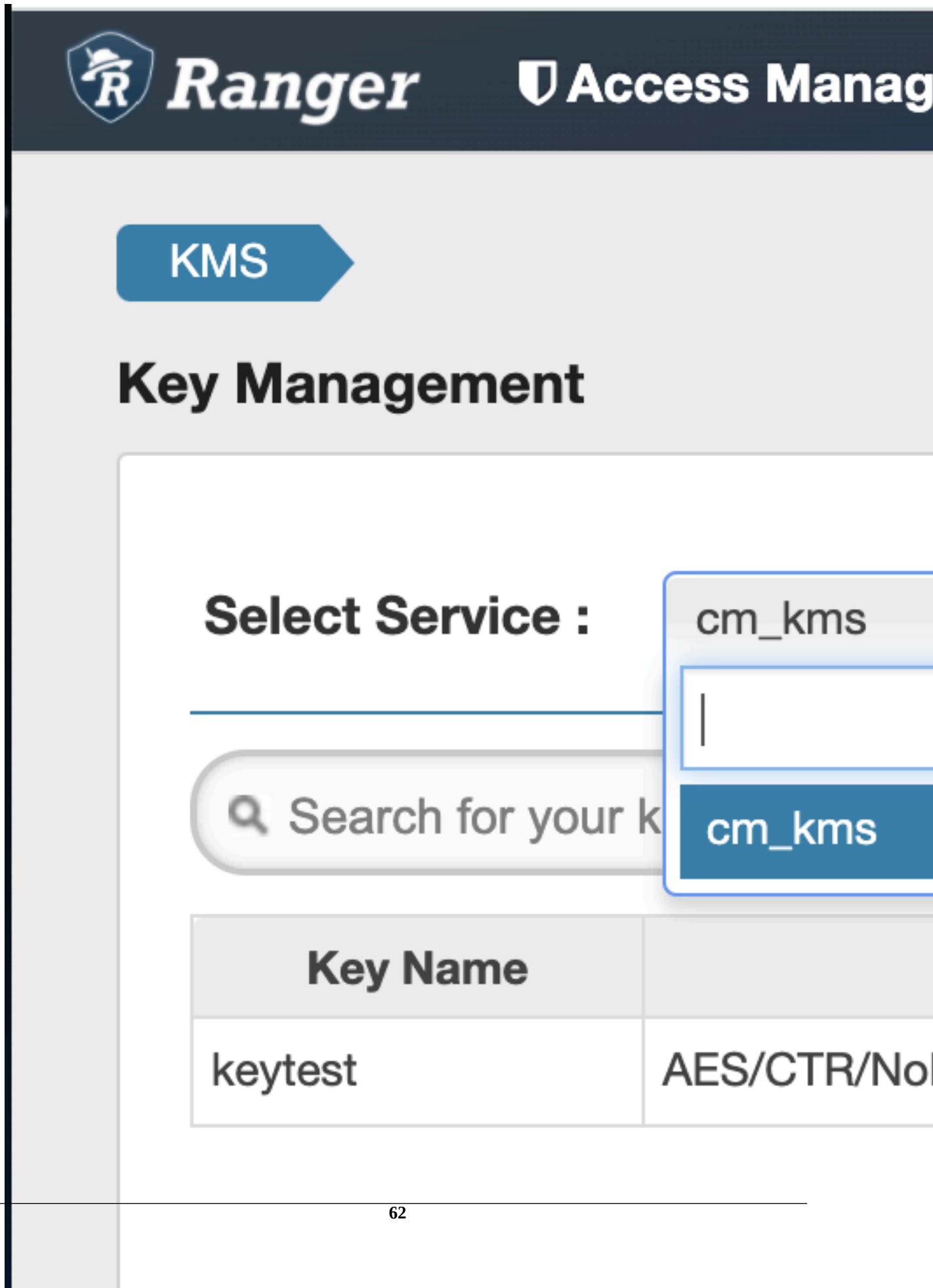

Ranger
Access Manag
Service Manager
Edit Service
Edit Service
Service Details :

List and Create Keys

How to list and create Ranger KMS keys.

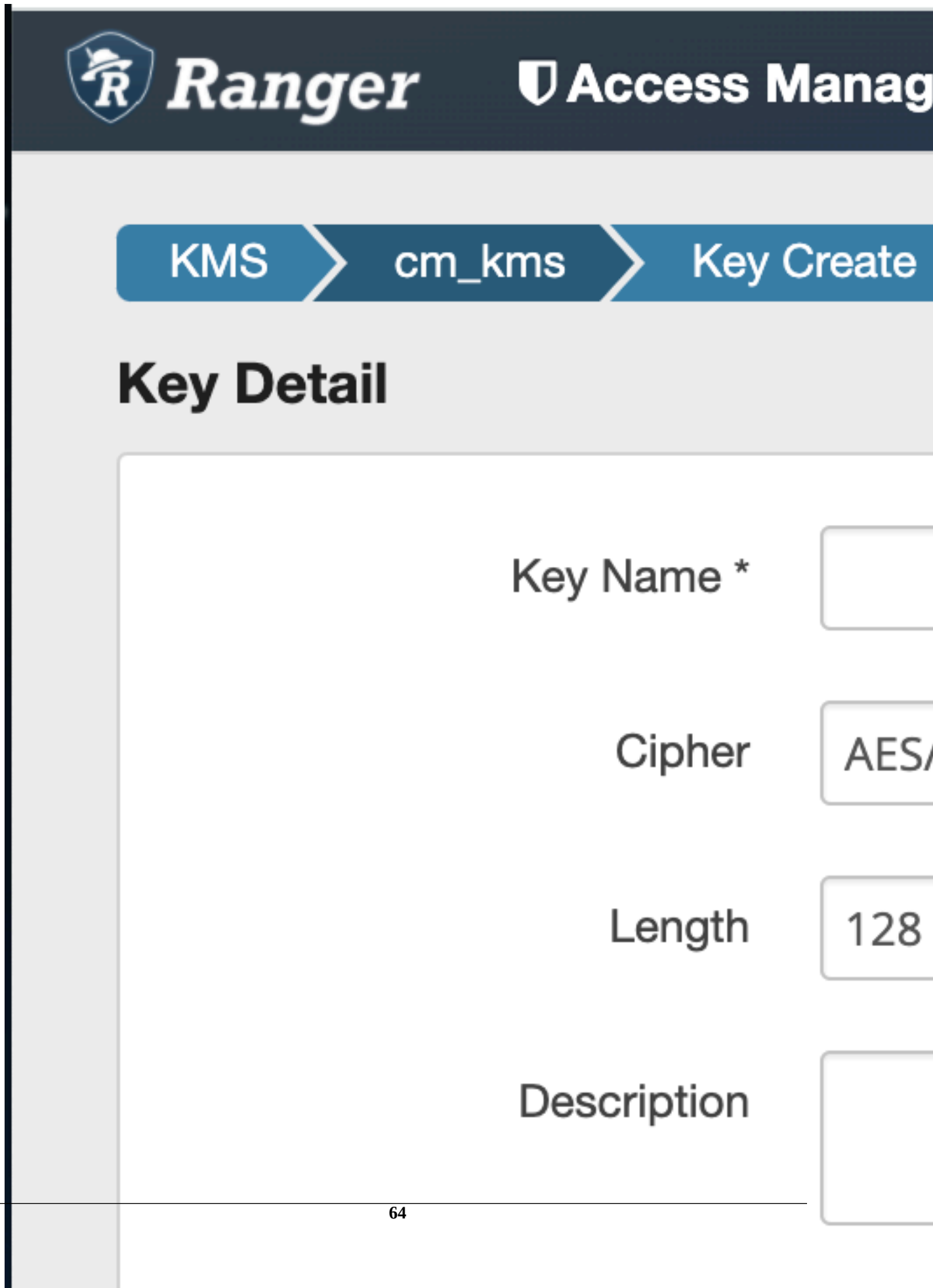
List existing keys

1. Log in to Ranger as the Ranger KMS admin user.
2. Click Encryption in the top menu to display the Key Management page.
3. Use the Select Service box to Select a Ranger KMS service. The keys for the service are dislayed.


Ranger
Access Manag
KMS
Key Management
Select Service :
cm_kms
Search for your k
cm_kms
Key Name
keytest
AES/CTR/No

Create a new key

1. Click Add New Key.
2. On the Key Detail page, add a valid key name.
3. Specify a cipher. Ranger KMS supports AES/CTR/NoPadding as the cipher suite.
4. Specify the key length: 128 or 256 bits.
5. Add other attributes as needed, then click Save.


Ranger
Access Manag
KMS
cm_kms
Key Create
Key Detail
Key Name *
Cipher
AES
Length
128
Description

Roll Over an Existing Key

How to roll over an existing Ranger KMS key.

About this task

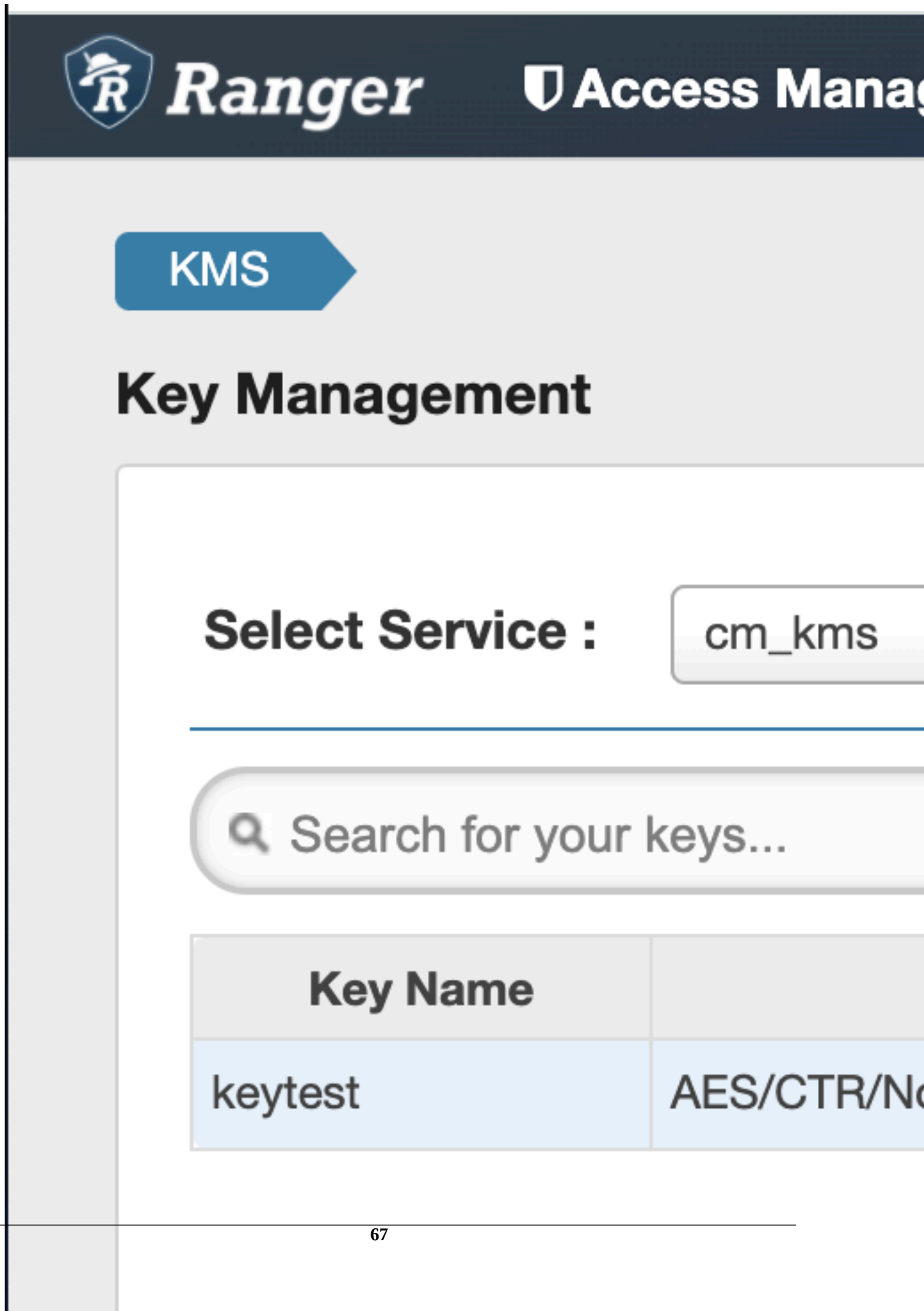
Rolling over (or "rotating") a key retains the same key name, but the key will have a different version. This operation re-encrypts existing file keys, but does not re-encrypt the actual file. Keys can be rolled over at any time.

After a key is rotated in Ranger KMS, new files will have the file key encrypted by the new master key for the encryption zone.

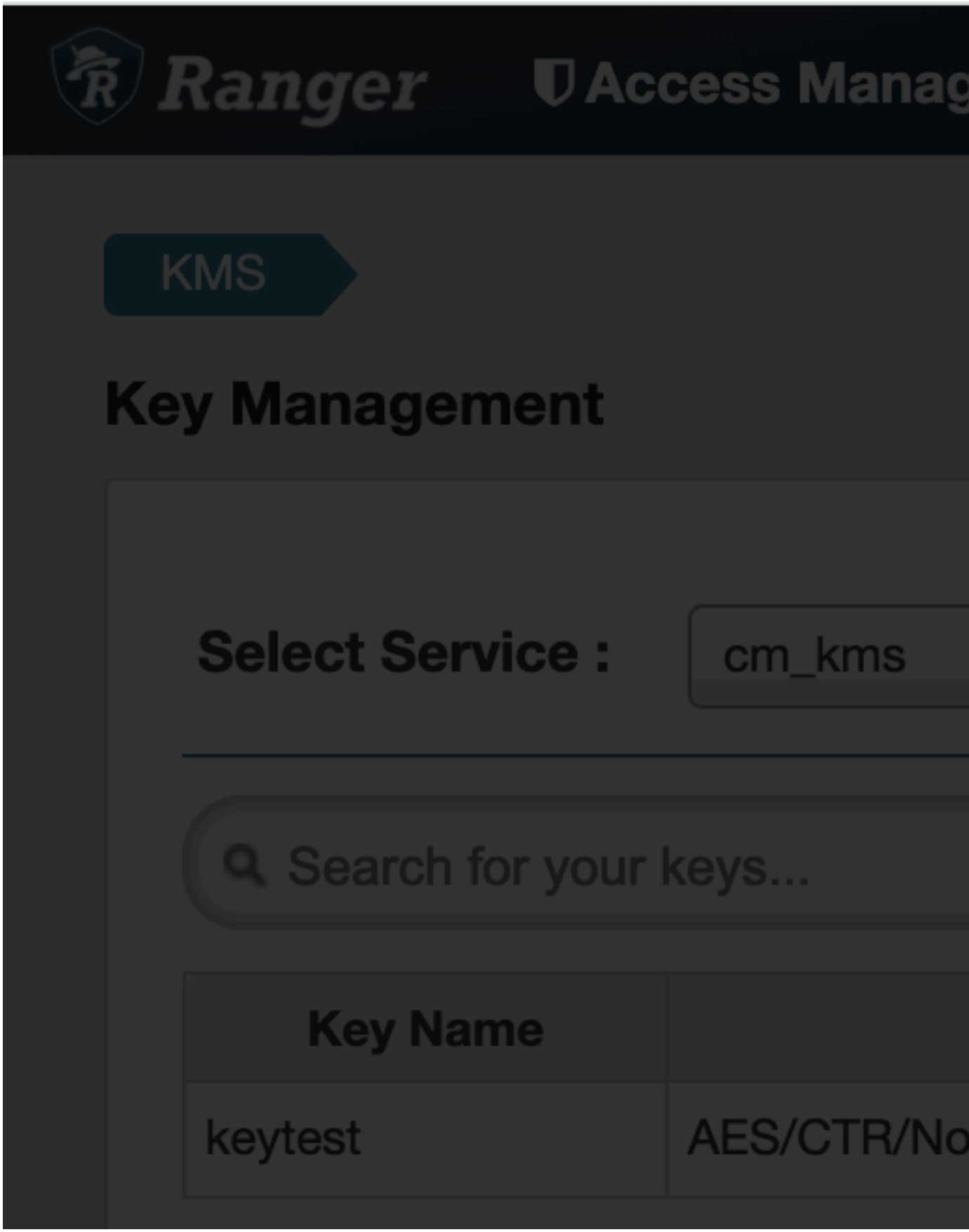
Procedure

1. Log in to Ranger as the Ranger KMS admin user, click Encryption in the top menu, then select a Ranger KMS service.

2. To rotate a key, click the Rollover icon for the key in the Action column.


Ranger
Access Manag
KMS
Key Management
Select Service :
cm_kms
Search for your keys...
Key Name
keytest
AES/CTR/N

3. Click OK on the confirmation pop-up.



Delete a Key

How to delete a Ranger KMS key.

About this task

Important:

Deleting a key associated with an existing encryption zone will result in data loss.

Procedure

1. Log in to Ranger as the Ranger KMS admin user, click Encryption in the top menu, then select a Ranger KMS service.
2. Click on the Delete icon for the key in the Action column.
3. Click OK on the confirmation pop-up.