Cloudera Manager

Encrypting Data at Rest in Cloudera Manager

Date published: 2024-12-10
Date modified: 2024-12-10

https://docs.cloudera.com/

Legal Notice

Contents

Encrypting Data at Rest

Secure data at rest using encryption mechanisms and key management.

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 Ranger KMS.

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

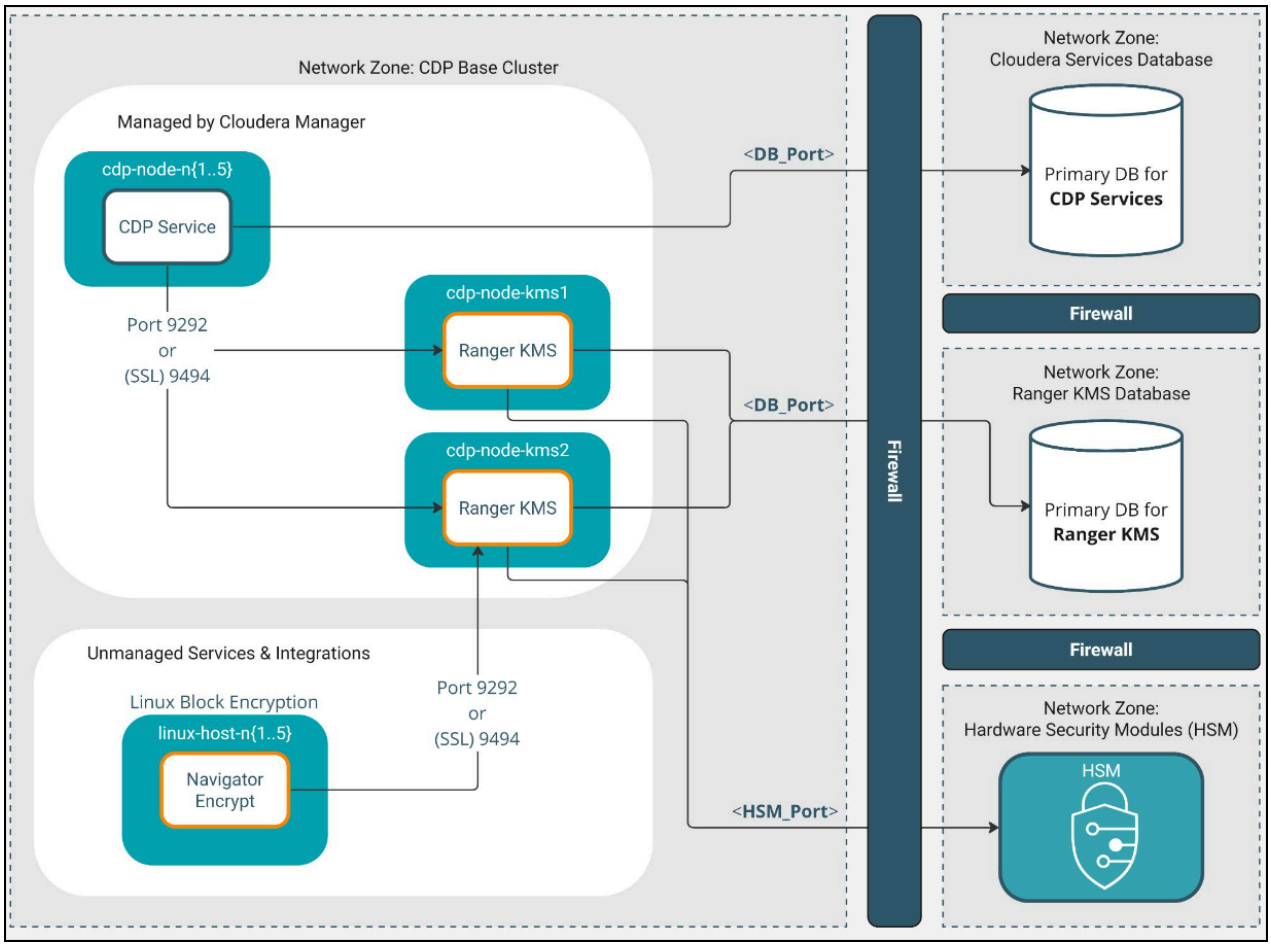
Related Information

Data at Rest Encryption Requirements

Data at Rest Encryption Reference Architecture

Encrypting Data at Rest - Deploying Ranger KMS with database

The following diagram illustrates product component functional relationships:



The Ranger KMS database is an external database.

For the Navigator Encrypt to Ranger KMS link, Kerberos authentication is mandatory. TLS is highly recommended.

Cloudera Manager supports multiple Ranger KMS instances, scaled horizontally, which provides High Availability.

For more details on Ranger KMS, see *Ranger KMS*.

Related Information

Resource Planning for Data at Rest Encryption
Ranger KMS overview

Data at Rest Encryption Requirements

Encryption comprises several components, each with its own requirements.

Overview

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.

Note: Due to a recent Linux kernel change the 'entropy_avail' reported by the kernel will always be 256. This will lead to erroneous alerts being issued if you changed the default settings for 'Host Entropy Thresholds'. These values will need to be changed to reflect the 256 value for 'entropy_avail'.

For RHEL 7, run the following commands:

```
sudo yum install rng-tools
cp /usr/lib/systemd/system/rngd.service /etc/systemd/system/
systemctl daemon-reload
systemctl start rngd
systemctl enable rngd
```

Make sure that the hosts running Ranger KMS, and Navigator Encrypt have sufficient entropy to perform cryptographic operations.

Ranger KMS Requirements

Recommended Hardware and Supported Distributions

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".

The Ranger 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 Ranger services to prevent the passage of plain text key material between the KMS and HDFS data nodes.

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 Ranger KMS installations, Navigator Encrypt initiates TCP traffic over port 9494 (HTTPS) to Ranger KMS.

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.

Network Time Protocol (NTP)

The Network Time Protocol (NTP) service synchronizes system time. Cloudera recommends using NTP to ensure that timestamps in system logs, cryptographic signatures, and other auditable events are consistent across systems.

Package Dependencies

Navigator Encrypt requires these packages, which are resolved by your distribution package manager during installation:

- dkms
- keyutils
- 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

Product Compatibility Matrix for Cloudera Navigator Encryption
Encrypting Data at Rest
Resource Planning for Data at Rest Encryption

Resource Planning for Data at Rest Encryption

High Availablity for Ranger KMS

For production environments, you must configure high availability for:

- Ranger KMS

Ranger KMS HA Planning

For high availability, you must provision at least two dedicated Ranger KMS hosts, for a minimum of four separate hosts. Do not run multiple Ranger 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 Ranger 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 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 Ranger KMS in its own dedicated cluster. See "Data at Rest Encryption Reference Architecture" for more information.

Virtual Machine Considerations

If you are using virtual machines, make sure that the resources (such as virtual disks, CPU, and memory) for each Ranger 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.

Important:

Each and every HDFS operation, even for non-encrypted data, will contact KMS. This impacts CPU requirements and must be considered when planning Key Management System implementations.

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 7 Security Guide

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

Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7 Download
Optimizing Performance for HDFS Transparent Encryption
Setting up Data at Rest Encryption for HDFS

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 Ranger KMS 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.

Data Encryption Components and Solutions

Cloudera supports two 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. 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.

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

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.

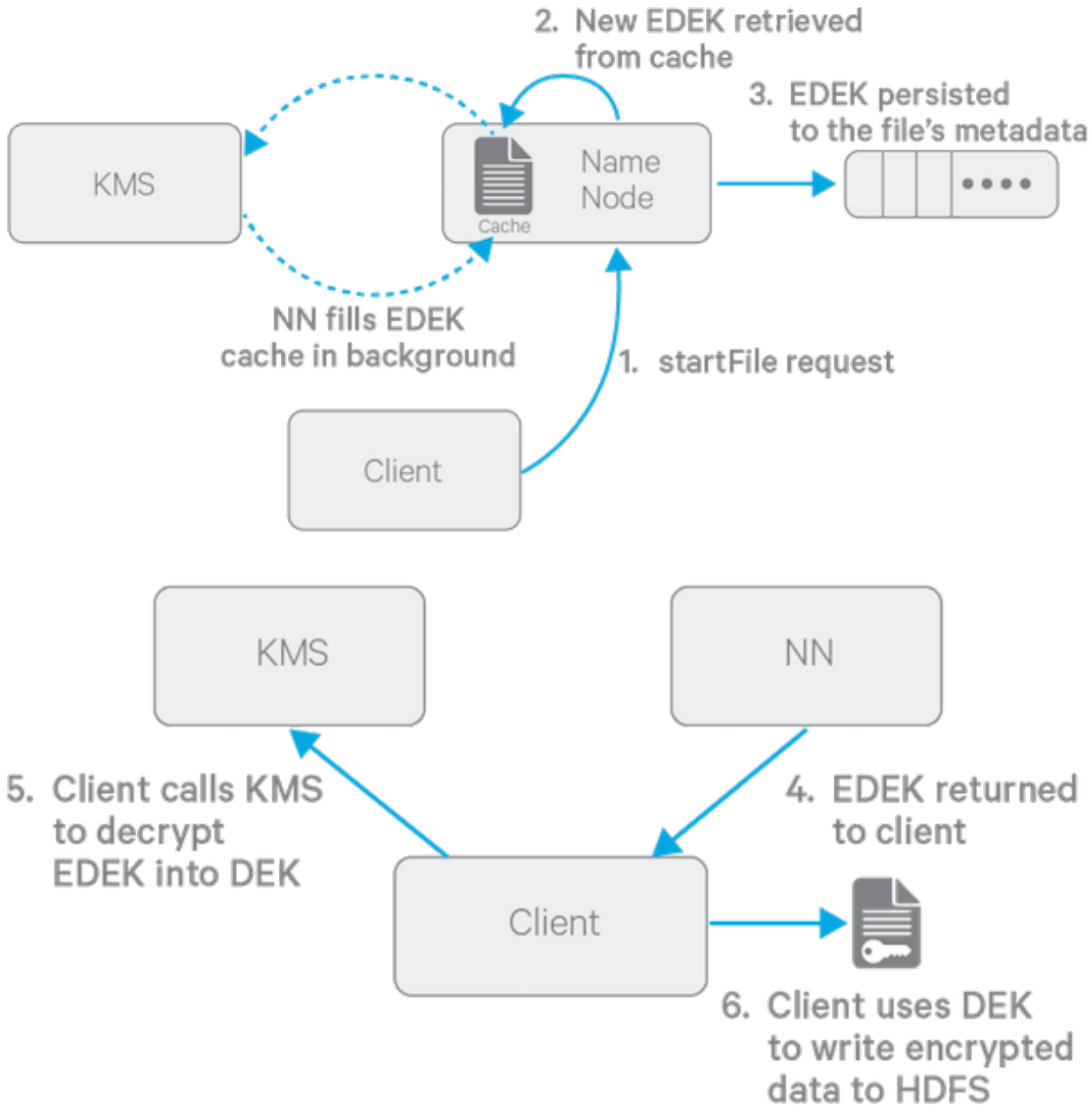
Related Information

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 HDFS client 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 NameNode provides the HDFS client with the file's EDEK and the version number of the EZ key that was used to generate the EDEK. The HDFS client 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.

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.

```
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):

  ```
  hadoop key create <key_name>
  ```

- Create a new empty directory and make it an encryption zone using the key created above.

  ```
  hadoop fs -mkdir /encryption_zone
  ```

```
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:

```
$ 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:

```
hadoop distcp /user/dir /encryption_zone
```

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

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.

```
hadoop fs -rm -r -skipTrash /encryption_zone
```

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)
}
```

Deleting Encryption Zone Keys

For information on how to delete encryption zone keys, see Delete a Key

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
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

Column Name	Description	Sample Data
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 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.

Note: Ranger audit directories in HDFS are owned by specific service users. In case of HDFS, it will be the HDFS service user. Typically, HDFS service user is blacklisted for DECRYPT_EEK operation. Hence, enabling encryption of ranger audit directories in HDFS is not recommended.

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: /warehouse/tablespace/

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 /warehouse/table space/ to /warehouse/tablespace-old, create an encryption zone at /warehouse/tablespace/, and then distcp all the data from /warehouse/tablespace-old to /warehouse/tablespace/.

In Cloudera Manager, configure the Hive Scratch Directory (hive.exec.scratchdir) to be inside the encryption zone by setting it to /warehouse/tablespace/tmp, ensuring that permissions are 1777 on /warehouse/tablespace/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_M AXSIZE. 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.

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.