HDP Configuring HDFS Encryption 3

Configuring Apache HDFS Encryption

Date of Publish: 2018-07-15

http://docs.hortonworks.com

Contents

HDFS Encryption

HDFS data at rest encryption implements end-to-end encryption of data read from and written to HDFS. End-to-end encryption means that data is encrypted and decrypted only by the client. HDFS does not have access to unencrypted data or keys.

Ranger KMS Administration

The Ranger Key Management Service (Ranger KMS) is a open source, scalable cryptographic key management service supporting HDFS "data at rest" encryption.

Ranger KMS is based on the Hadoop KMS originally developed by the Apache community. The Hadoop KMS stores keys in a file-based Java keystore by default. Ranger extends the native Hadoop KMS functionality by allowing you to store keys in a secure database.

Ranger provides centralized administration of the key management server through the Ranger admin portal.

There are three main functions within the Ranger KMS:

- Key management. Ranger admin provides the ability to create, update or delete keys using the Web UI or REST APIs. All Hadoop KMS APIs work with Ranger KMS using the keyadmin username and password.
- Access control policies. Ranger admin also provides the ability to manage access control policies within Ranger KMS. The access policies control permissions to generate or manage keys, adding another layer of security for data encrypted in Hadoop.
- Audit. Ranger provides full audit trace of all actions performed by Ranger KMS.

Ranger KMS along with HDFS encryption are recommended for use in all environments. In addition to secure key storage using a database, Ranger KMS is also scalable, and multiple versions of Ranger KMS can be run behind a load balancer.

Related Information

HDFS "Data at Rest" Encryption

Hadoop KMS APIs

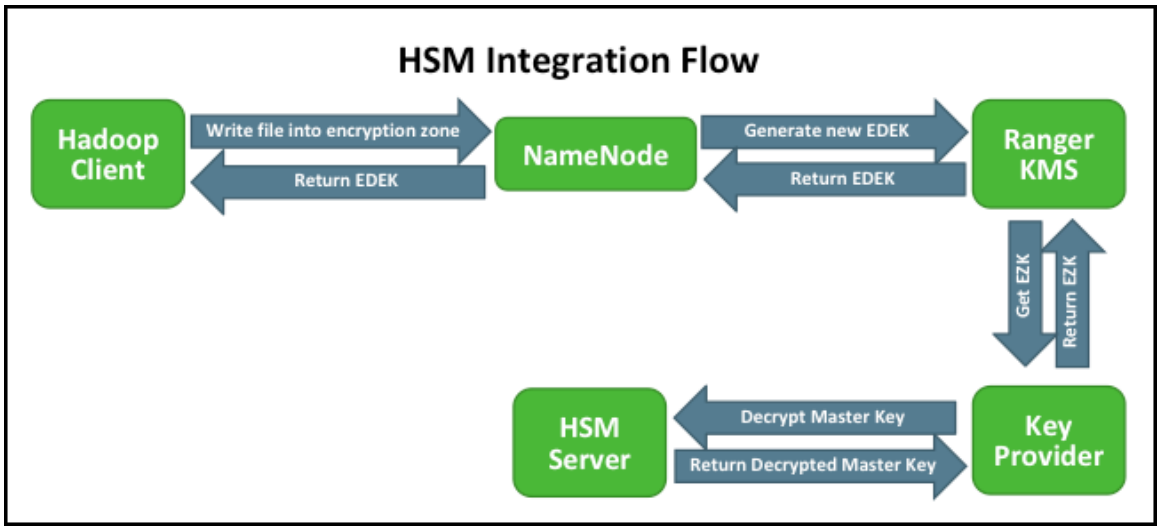
Store Master Key in a Hardware Security Module (HSM)

PCI compliance requires that keys are stored in Hardware Security Modules (HSMs) rather than a software KMS. For example, this is required for financial institutions working with customer credit/debit card terminals. This section explains how to store keys in an HSM.

Before you begin

You must have a separate partition for each KMS cluster.

About this task



Related Information

PCI Compliance

Installing Ranger KMS Hardware Security Module (HSM)

You can install the Ranger KMS HSM in three ways: manually, via Ambari with a plain text password, or via Ambari with JCEKS.

You must have a separate partition for each KMS cluster.

Related Information

Install the SafeNet Luna SA Client Software

Install Ranger KMS HSM Manually

How to install the Ranger KMS HSM manually.

Before you begin

- Install the SafeNet Luna SA Client software (link below).
- You must have a separate partition for each KMS cluster.

Procedure

Refer to the instructions on the Apache Wiki (link below).

Related Information

Install the SafeNet Luna SA Client Software

Apache Wiki>Installing Ranger KMS HSM (Manually)

Install Ranger KMS HSM via Ambari with plain text password

How to install the Ranger KMS HSM via Ambari with a plain text password.

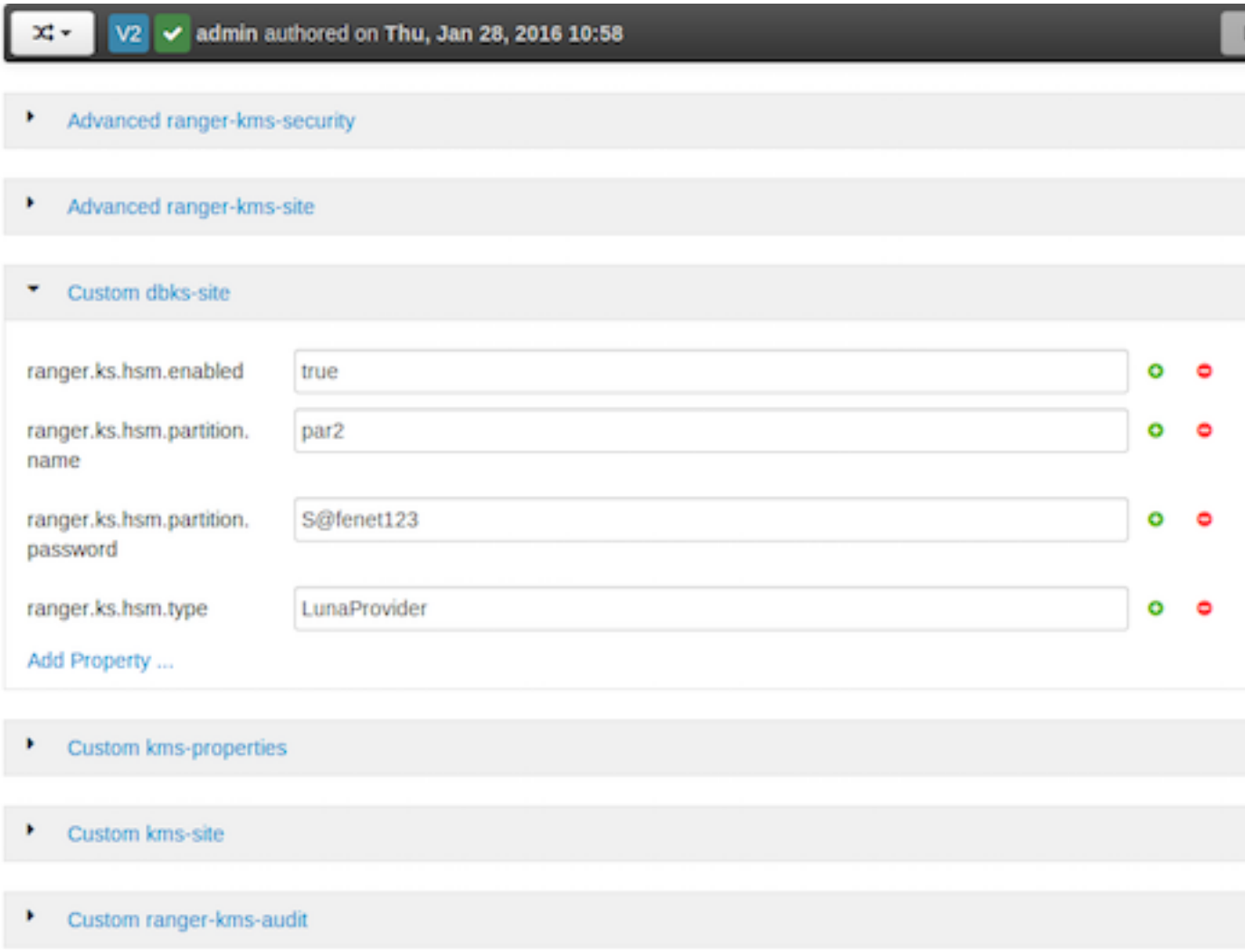
Before you begin

- Install the SafeNet Luna SA Client software (link below).
- You must have a separate partition for each KMS cluster.

Procedure

1. Complete “Installing the Ranger Key Management Service”.
2. While configuring add the HSM related properties in Advanced dbks-site Menu (dbks-site.xml):

- ranger.ks.hsm.enabled=true
- ranger.ks.hsm.partition.name=Partition Name
- ranger.ks.hsm.partition.password=Partition Password
- ranger.ks.hsm.type=LunaProvider



3. Click on **Next** and follow the instructions to install Ranger KMS.

Related Information

Installing the Ranger Key Management Service
Install the SafeNet Luna SA Client Software

Install Ranger KMS HSM via Ambari with JCEKS

How to install the Ranger KMS HSM via Ambari with JCEKS.

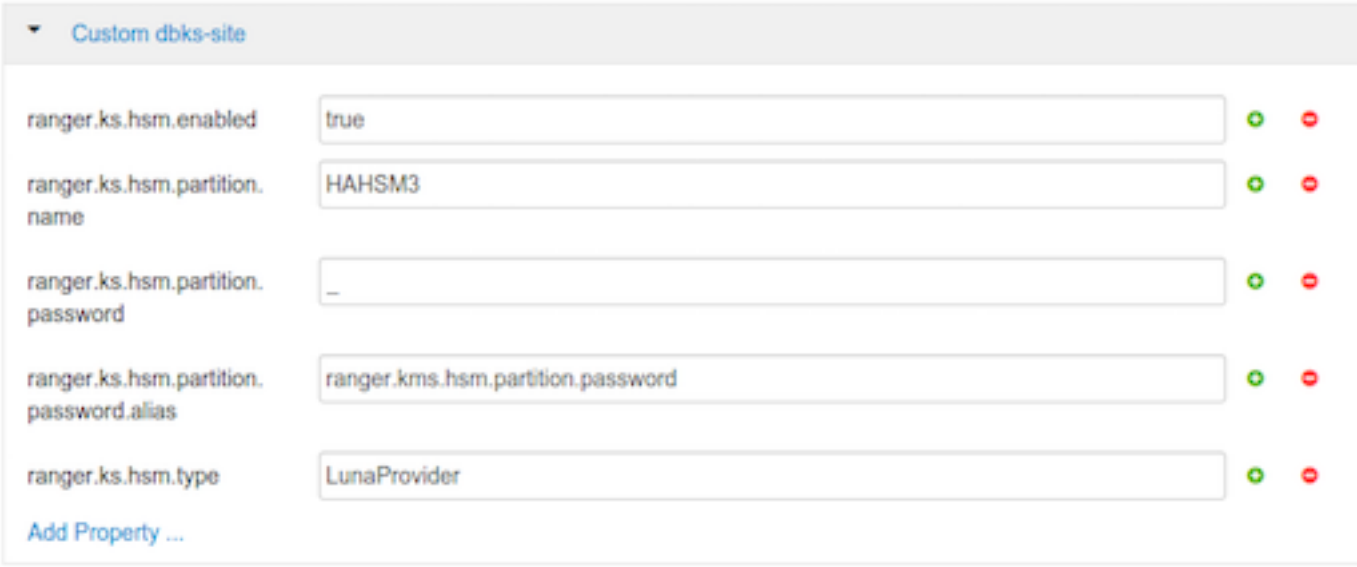
Before you begin

- Install the SafeNet Luna SA Client software (link below).
- You must have a separate partition for each KMS cluster.

Procedure

1. Complete “Installing the Ranger Key Management Service”.
2. While configuring add the HSM related properties in Advanced dbks-site Menu (dbks-site.xml):

- ranger.ks.hsm.enabled=true
- ranger.ks.hsm.partition.name=Partition Name
- ranger.ks.hsm.partition.password=_
- ranger.ks.hsm.partition.password.alias=ranger.kms.hsm.partition.password
- ranger.ks.hsm.type=LunaProvider



3. Click on **Next** and follow the instructions to install Ranger KMS.
   Ranger KMS will fail to start (expected behavior).
4. Execute this command on the cluster where Ranger KMS is installed:

```
python /usr/hdp/current/ranger-kms/ranger_credential_helper.py -l "/usr/
hdp/current/ranger-kms/cred/lib/*" -f /etc/ranger/kms/rangerkms.jceks -k
 ranger.kms.hsm.partition.password -v <Partition_Password> -c 1
```

5. Restart the KMS from Ambari.

Related Information

Installing the Ranger Key Management Service
Install the SafeNet Luna SA Client Software

Configure HSM for High Availability (HA)

How to configure HSM for high availability.

Before you begin

You must have at least two Luna SA appliances with PED Authentication, or two with Password Authentication.

Procedure

1. Set up appliances for HA:
   a) Perform the network setup on both HA units: install the SafeNet Luna SA Client software (link below).
   b) In hsm showPolicies, ensure that Allow Cloning=on and Allow Network Replication=on.
   c) Initialize the HSMs on your Luna SA appliances. They must have the same cloning domain (i.e., must share the same red, domain PED Key if they are PED-authenticated) or they must share the same domain string if they are password-authenticated.

d) Create a partition on each Luna SA. They do not need to have the same labels, but must have the same password.
e) Record the serial number of each partition created on each Luna SA (use partition show).

2. Register clients with Luna SA HA:

a) Proceed with normal client setup, “Prepare the Client for Network Trust Link” (link below).
b) Register your client computer with both Luna SAs.
c) Verify using ./vtl verify command. It should show the numbers of partitions registered with client.

3. Create the HA GroupNote for your client version:

- Version 5
- Version 6

4. Version 5

a) After creating partitions on (at least) two Luna appliances, and setting up Network Trust Links between those partitions and your client, use LunaCM to configure HA on your client: Go to the directory: /usr/safenet/lunaclient/bin/.
b) To add members in haadmin, create a new group on the client: ./vtl haAdmin newGroup -serialNum HA Group Number -label Groupname -password password .
For example: ./vtl haAdmin newGroup -serialNum 156453092 -label myHAgroup -password S@fenet123
c) Add members into your haadmin: ./vtl haAdmin addMember -group HA Group Number -serialNum serial_number -password password .
For example: ./vtl haAdmin addMember -group 1156453092 -serialNum 156451030 -password S@fenet123
d) Enable synchronization of HAadmin Members: ./vtl haAdmin synchronize -group HA Group Number -password password .
For example: ./vtl haAdmin synchronize -enable -group 1156453092 -password S@fenet123
e) To Enable HAOnly: ./vtl haAdmin HAOnly -enable.
f) Check haadmin status after synchronization: ./vtl haAdmin show.

> **Note:** After synchronization please verify kms master key copied to both partitions registered in hsm ha group. It takes time to copy master key to another partition.

5. Version 6

a) After creating partitions on (at least) two Luna appliances, and setting up Network Trust Links between those partitions and your client, use LunaCM to configure HA on your client:

1. Go to directory: /usr/safenet/lunaclient/bin/.
2. Select Lunacm: ./lunacm.

b) To add members in hagroup, create a new group on the client: haGroup creategroup -serialNumber serial number -l label -p password .
For example: lunacm:>haGroup creategroup -serialNumber 1047740028310 -l HAHSM3 -p S@fenet123
c) Use the hagroup addmember command to add new member into hagroup client: hagroup addMember -group groupname -serialNumber serial number -password password .

Field descriptions:

- Label for the group (do NOT call the group just "HA"): groupname
- The serial number of the first partition OR the slot number of the first partition: serial number
- The password for the partition: password
- Lunacm also generates and assigns a Serial Number to the group itself.

For example: lunacm:>hagroup addMember -group rkmsgroup -serialNumber 1047749341551 -password S@fenet123

d) Use the hagroup addmember command to add another member to the HA group: hagroup addMember -group groupname -serialNumber serial number -password password .

For example: lunacm:>hagroup addMember -serialNumber 1047740028310 -g rkmslgroup -password S@fenet123

e) Check group member in group using "hagroup listGroups" command: hagroup listGroups.

f) Enable HAOnly: hagroup HAOnly -enable.

g) Enable synchronization of HAgroup Members: hagroup synchronize -group groupname -password password -enable.
For example: lunacm:>hagroup synchronize -group rkmslgroup -password S@fenet123 -enable

6. After configuring HSM HA, to run Ranger KMS in HSM HA mode you must specify the virtual group name created above in HSM_PARTITION_NAME property of install.properties and setup and start Ranger KMS. Note: All other configuration for HSM in install.properties of Ranger KMS as mentioned in “Installing Ranger KMS HSM” will remain the same.

Related Information

Install the SafeNet Luna SA Client Software

Prepare the Client for Network Trust Link

Migrate between HSM and Ranger DB

If required, you can migrate from HSM to Ranger DB or Ranger DB to HSM.

Procedure

1. If running, stop the Ranger KMS server.
2. Go to the Ranger KMS directory: /usr/hdp/$version/ranger-kms.

   DB details must be correctly configured to which KMS needs migration to (located in the xml config file of Ranger KMS).

   For DB to HSM: HSM details must be the KMS HSM to which we are migrating.
3. Run:

Option	Run	Example
DB to HSM	./DBMK2HSM.sh $provider $HSM_PARTITION_NAME	./DBMK2HSM.sh LunaProvider par19
HSM to DB	./HSMMK2DB.sh $provider $HSM_PARTITION_NAME	./HSMMK2DB.sh LunaProvider par19

4. Enter the partition password.
5. After the migration is completed: if you want to run Ranger KMS according to the new configuration (either with HSM enabled or disabled,) update the Ranger KMS properties if required.
6. Start Ranger KMS from Ambari.

What to do next

Warning:

Deleting the master key is a destructive operation. If the master key is lost, there is potential data loss, since data under encryption zones cannot be recovered. Therefore, it is a best practice to keep backups of the master key in DB as well as HSM.

- DB to HSM: When Ranger KMS is running with HSM enabled: from DB table “ranger_masterkey”, delete the Master Key row if it is not required as Master Key already being migrated to HSM.
- HSM to DB: When Ranger KMS is running with HSM disabled: from HSM, clear the Master Key object from the partition if it is not required as Master Key already being migrated to DB.

Optional: Clear Objects from the HSM Partition

How to clear objects from the HSM partition.

Procedure

1. SSH to the HSM Appliance Server.
   ssh admin@elab6.safenet-inc.com
2. Enter Password for the HSM Appliance Server when prompted.
3. Check the Partition Objects that you want to clear and enter the password for the partition when prompted: Partition showContents -par partition_name .
   partition showContents -par par14

   Note:

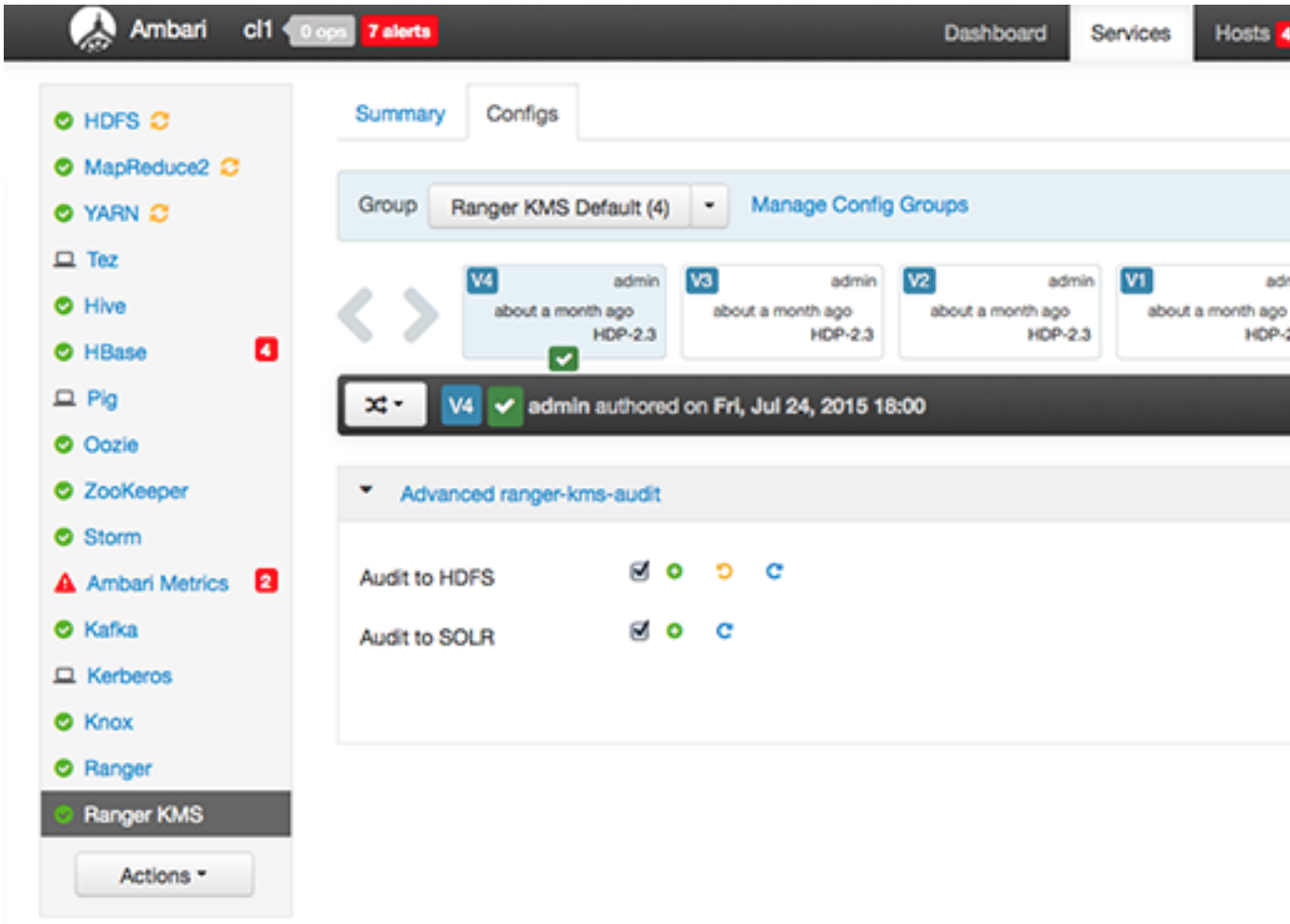
   All objects listed will be destroyed during step 3.
4. Clear the objects from HMS partition: Partition clear -par partition_name .
5. Enter Password for Partition when prompted.
   partition clear -par par14

Enable Ranger KMS Audit

Ranger KMS supports audit to DB, HDFS, and Solr. Solr is well-suited for short-term auditing and UI access (for example, one month of data accessible via quick queries in the Web UI). HDFS is typically used for archival auditing. They are not mutually exclusive; we recommend configuring audit to both Solr and HDFS. First, make sure Ranger KMS logs are enabled by following these steps.

Procedure

1. Go to the Ambari UI: http://<gateway>:8080.
2. Select ranger-kms from the service.
3. Click the Configs tab, and go to the accordion menu.
4. In the Advanced ranger-kms-audit list, set xasecure.audit.is.enabled to true.
5. Select "Audit to Solr" and/or "Audit to HDFS", depending on which database(s) you plan to use:



6. Save the configuration and restart the Ranger KMS service.
7. Check to see if the Ranger KMS Plugin is enabled:
   a) Go to the Ranger UI: http://<gateway>:6080.
   b) Login with your keyadmin user ID and password (the defaults are keyadmin, keyadmin). The default repository will be added under KMS service.
   c) Run a test connection for the service. You should see a ‘connected successfully’ pop-up message. If the connection is not successful, make sure that the configured user exists (in KDC for a secure cluster).
   d) Choose the Audit > Plugin tab.
   e) Check whether plugins are communicating. The UI should display Http Response code 200 for the respective plugin.

Save Audits to Solr

How to save audits to Solr, when enabling Ranger KMS Audit.

Procedure

1. From the Ambari dashboard, select the Ranger service. Select Configs > Advanced, then scroll down and select Advanced ranger-admin-site. Set the following property value: ranger.audit.source.type = solr.
2. On the Ranger Configs tab, select Ranger Audit. The SolrCloud button should be set to ON. The SolrCloud configuration settings are loaded automatically when the SolrCloud button is set from OFF to ON, but you can also manually update the settings.

3. Restart the Ranger service.
4. Next, to enable Ranger KMS auditing to Solr, set the following properties in the Advanced ranger-kms-audit list:
   a) Check the box next to Enable audit to solr in the Ranger KMS component.
   b) Check the Audit provider summary enabled box, and make sure that xasecure.audit.is.enabled is set to true.
   c) Restart Ranger KMS.

Save Audits to HDFS

How to save audits to HDFS, when enabling Ranger KMS Audit.

About this task

There are no configuration changes needed for Ranger properties.

To save Ranger KMS audits to HDFS, set the following properties in the Advanced ranger-kms-audit list.

Note: The following configuration settings must be changed in each Plugin.

Procedure

1. Check the box next to Enable Audit to HDFS in the Ranger KMS component.
2. Set the HDFS path to the path of the location in HDFS where you want to store audits: xasecure.audit.destination.hdfs.dir = hdfs://NAMENODE_FQDN:8020/ranger/audit.
3. Check the Audit provider summary enabled box, and make sure that xasecure.audit.is.enabled is set to true.
4. Make sure that the plugin's root user (kms) has permission to access HDFS Path hdfs://NAMENODE_FQDN:8020/ranger/audit.
5. Restart Ranger KMS.
6. Generate audit logs for the Ranger KMS.
7. (Optional) To verify audit to HDFS without waiting for the default sync delay (approximately 24 hours), restart Ranger KMS. Ranger KMS will start writing to HDFS after the changes are saved post-restart.
8. To check for audit data: hdfs dfs -ls /ranger/audit/.
9. Test Ranger KMS audit to HDFS:
   a) Under custom core-site.xml, set hadoop.proxyuser.kms.groups to “*” or to the service user.
   b) In the custom kms-site file, add hadoop.kms.proxyuser.keyadmin.users and set its value to "*". (If you are not using keyadmin to access Ranger KMS Admin, replace “keyadmin” with the user account used for authentication.)
   c) In the custom kms-site file, add hadoop.kms.proxyuser.keyadmin.hosts and set its value to "*". (If you are not using keyadmin to access Ranger KMS Admin, replace “keyadmin” with the user account used for authentication.)
   d) Choose:
      - Copy the core-site.xml to the component’s class path (/etc/ranger/kms/conf)
      - Link to /etc/hadoop/conf/core-site.xml under /etc/ranger/kms/conf (ln -s /etc/hadoop/conf/core-site.xml /etc/ranger/kms/conf/core-site.xml)
   e) Verify the service user principal. (For Ranger KMS it will be the http user.)
   f) Make sure that the component user has permission to access HDFS. (For Ranger KMS the http user should also have permission.)

Enable SSL for Ranger KMS

How to enable SSL for Ranger KMS. If you do not have access to Public CA-issued certificates, complete the following steps to create and configure self-signed certificates.

About this task

Considerations:

- Copy keystore/truststore files into a different location (e.g. /etc/security/serverKeys) than the /etc/<component>/conffolders.
- Make sure JKS file names are different from each other.
- Make sure correct permissions are applied.
- Make sure all passwords are secured.
- For the test connection to be successful after enabling SSL, self-signed certificates should be imported to the Ranger admin's trust store (typically JDK cacerts).
- Property ranger.plugin.service.policy.rest.ssl.config.file should be verified; for example:

  ranger.plugin.kms.policy.rest.ssl.config.file ==> /etc/ranger/kms/conf/ranger-policymgr-ssl.xml

Procedure

1. Stop the Ranger KMS service:

2. Go to the Ranger KMS (and plugin) installation location, and create a self-signed certificate:

```
cd /etc/ranger/kms/conf/
keytool -genkey -keyalg RSA -alias rangerKMSAgent -keystore <ranger-kms-
ks> -storepass myKeyFilePassword -validity 360 -keysize 2048
chown kms:kms <ranger-kms-ks>
chmod 400 <ranger-kms-ks>
```

   where

   <ranger-kms-ks> is the name of the Ranger KMS keystore (for example, ranger-plugin-keystore.jks)

3. Provide an identifiable string in response to the question "What is your first and last name?"

   Important: In case multiple servers need to communicate with Ranger admin for downloading policies for the same service/repository, make sure to use the repo name or a common string across all nodes. Remember exactly what you entered, because this value will be required for the Common Name for Certificate field on the edit repository page in the policy manager UI.

   To create the keystore, provide answers to the subsequent questions. Note: Press enter when prompted for a password.

4. Create a truststore for the Ranger KMS plugin, and add the public key of admin as a trusted entry into the truststore:

```
cd /etc/ranger/kms/conf/

keytool -export -keystore <ranger-admin-ks> -alias rangeradmin -file
 <cert-filename>
```

```
keytool -import -file <cert-filename> -alias rangeradmintrust -keystore
 <ranger-kms-ts> -storepass changeit

chown kms:kms <ranger-kms-ts>

chmod 400 <ranger-kms-ts>
```

where

<ranger-admin-ks> is the location of the Ranger Admin keystore (for example, /etc/ranger/admin/conf/ranger-admin-keystore.jks)

<ranger-kms-ts> is the name of the Ranger KMS plugin trustore (for example, ranger-plugin-truststore.jks)

<cert-filename> is the name of the Ranger Admin certificate file (for example, ranger-admin-trust.cer)

Note: Press enter when prompted for a password.

5. Update below properties available in Advanced ranger-kms-policymgr-ssl:
   a) xasecure.policymgr.clientssl.keystore: Provide the location for the keystore that you created in the previous step.
   b) xasecure.policymgr.clientssl.keystore.password: Provide the password for the keystore (myKeyFilePassword).
   c) xasecure.policymgr.clientssl.truststore: Provide the location for the truststore that you created in the previous step.
   d) xasecure.policymgr.clientssl.truststore.password: Provide the password for the truststore (changeit).
6. Add the plugin's self-signed cert into Admin's trustedCACerts:

```
cd /etc/ranger/admin/conf
keytool -export -keystore <ranger-kms-ks> -alias rangerKMSAgent -file
 <cert-filename> -storepass myKeyFilePassword
keytool -import -file <cert-filename> -alias rangerkmsAgentTrust -keystore
 <ranger-admin-ts> -storepass changeit
```

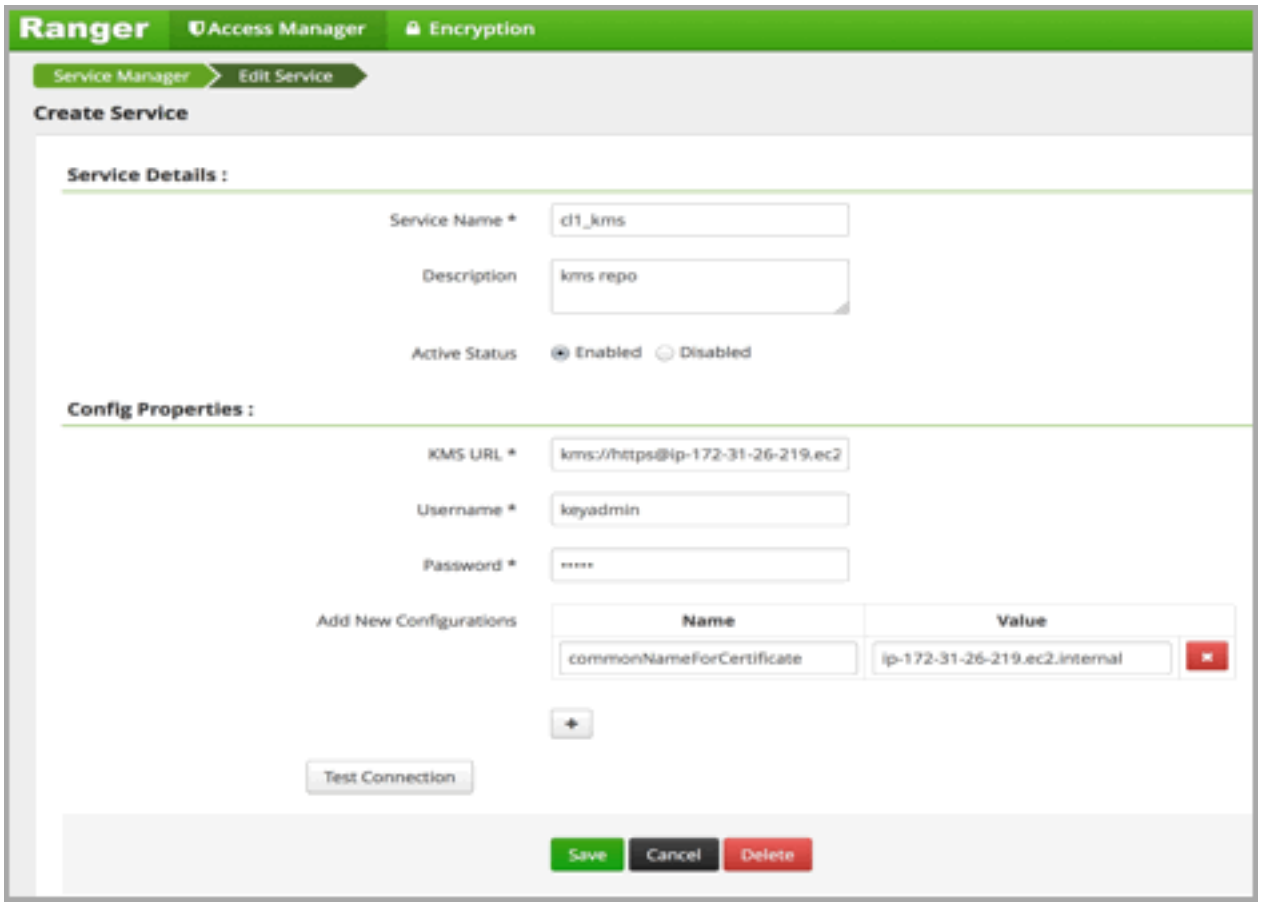
where

<ranger-kms-ks> is the path to the Ranger KMS keystore (for example, /etc/ranger/kms/conf/ranger-plugin-keystore.jks)

<cert-filename> is the name of the certificate file (for example, ranger-kmsAgent-trust.cer)

<ranger-admin-ts> is the name of the Ranger Admin truststore file (for example, the JDK cacerts file)

7. Log into the Policy Manager UI (as keyadmin user) and click on the Edit button of your KMS repository. Provide the CN name of the keystore for Common Name For Certificate (commonNameForCertificate), and save it. This property is not added by default.



8. Configure the Ranger KMS Server:
   a) Go to the Ranger KMS config location and create a self-signed certificate:

```
cd /etc/ranger/kms/conf
keytool -genkey -keyalg RSA -alias rangerkms -keystore <ranger-kms-ks> -
storepass rangerkms -validity 360 -keysize 2048
chown kms:kms ranger-kms-keystore.jks
chmod 400 ranger-kms-keystore.jks
```

   where

   <ranger-kms-ks> is the name of the Ranger KMS keystore (for example, ranger-plugin-keystore.jks)

   Provide an identifiable string in response to the question "What is your first and last name?" To create the keystore, provide answers to all subsequent questions to create the keystore Note: Press enter when prompted for a password.

   b) Edit the following properties and values in Advanced ranger-kms-site:

   - ranger.service.https.attrib.keystore.file: Add file path of ranger-kms-keystore.jks
   - ranger.service.https.attrib.client.auth: want
   - ranger.service.https.attrib.keystore.keyalias: Add the alias used for creating ranger-kms-keystore.jks
   - ranger.service.https.attrib.keystore.pass: Add password used for creating ranger-kms-keystore.jks
   - ranger.service.https.attrib.ssl.enabled: true

   c) Update kms_port in Advanced kms-env to 9393. Ambari will recommend the value to {{ranger.service.https.port}}.
   d) Save your changes and start Ranger KMS.

e) In your browser (or from Curl) when you access the Ranger KMS UI using the HTTPS protocol on the ranger.service.https.port listed in Ambari, the browser should respond that it does not trust the site. Proceed, and you should be able to access Ranger KMS on HTTPS with the self-signed cert that you just created.

f) Export the Ranger KMS certificate:

```
cd /usr/hdp/<version>/ranger-kms/conf
keytool -export -keystore <ranger-kms-ks> -alias rangerkms -file <cert-
filename>
```

where

<ranger-kms-ks> is the name of the Ranger KMS keystore (for example, ranger-kms-keystore.jks)

<cert-filename> is the name of the certificate file (for example, ranger-kms-trust.cer)

g) Import the Ranger KMS certificate into the Ranger admin truststore: keytool -import -file <cert-filename> -alias rangerkms -keystore <ranger-admin-ts> -storepass changeit.

where

<cert-filename> is the name of the certificate file (for example, ranger-kms-trust.cer)

<ranger-admin-ts> is the name of the Ranger Admin truststore file (for example, JDK cacerts)

Note:

Make sure Ranger Admin's truststore properties (ranger.truststore.file and ranger.truststore.password) are correctly configured in ranger-admin-site.xml.

h) Import the Ranger KMS certificate into the Hadoop client truststore: keytool -import -file <cert-filename> -alias rangerkms -keystore <ts-filename> -storepass bigdata.

where

<cert-filename> is the name of the certificate file (for example, ranger-kms-trust.cer)

<ts-filename> is the name of Hadoop client truststore file (for example, /etc/security/clientKeys/all.jks)

i) Restart Ranger Admin and Ranger KMS.

j) Login to Policy Manager UI with keyadmin credentials. Under default KMS Repo configuration, replace KMS URL configuration value with the new SSL-enabled KMS URL.

Previous KMS URL = kms://http@internal host name:http_port/kms

New KMS URL = kms://https@internal host name:https_port/kms

k) Now in the Policy Manager UI>Audit>Plugin tab, you should see an entry for your service name with HTTP Response Code = 200.

Install Multiple Ranger KMS

Multiple services can be set up for high availability of Ranger KMS. HDFS interacts with the active process. Follow these steps to install Ranger KMS on multiple nodes.

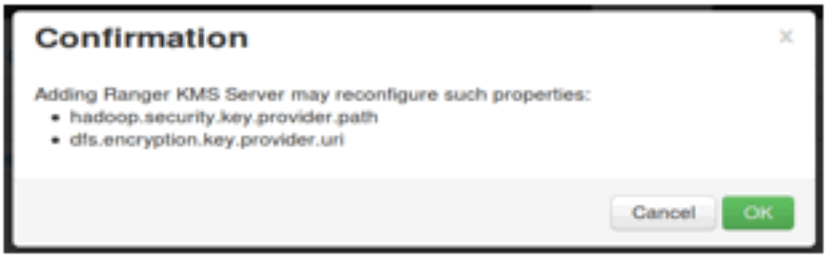
Before you begin

An instance with more than one node.

Procedure

1. First install Ranger KMS on a single node (see "Installing the Ranger Key Management Service").
2. Next, add the Ranger KMS service to another node. In the Ambari Web UI for the additional node, go to Ranger KMS service → Summary → Service Actions → Add Ranger KMS server.

3. After adding Ranger KMS server, Ambari will show a pop-up message.
4. Press OK. Ambari will modify two HDFS properties, hadoop.security.key.provider.path and dfs.encryption.key.provider.uri.
5. Restart the HDFS service:



6. For the Ranger KMS service, go to the Advanced kms-site list and change the following property values:

```
hadoop.kms.cache.enable=false
hadoop.kms.cache.timeout.ms=0
hadoop.kms.current.key.cache.timeout.ms=0
hadoop.kms.authentication.signer.secret.provider=zookeeper
hadoop.kms.authentication.signer.secret.provider.zookeeper.connection.string={intern
ip of first node}:2181,{internal ip of second node}:2181, ...
hadoop.kms.authentication.signer.secret.provider.zookeeper.auth.type=none
```

7. Save your configuration changes and restart the Ranger KMS service.
8. Next, check connectivity from Ranger admin for the newly-added Ranger KMS server:
   a) Go to the Ranger UI: http://<gateway>:6080.
   b) Login with your keyadmin user ID and password (the defaults are keyadmin, keyadmin; these should be changed as soon as possible after installation). The default repository will be added under Ranger KMS service.
   c) Under Config properties of the Ranger KMS URL, add the newly added Ranger KMS server FQDN. For example:

      Previous Ranger KMS URL = kms://http@<internal host name>:9292/kms

      New Ranger KMS URL = kms://http@<internal host name1>;<internal host name2>;...:9292/kms
   d) Run a test connection for the service. You should see a ‘connected successfully’ message.
   e) Choose the Audit > Plugin tab.
   f) Check whether plugins are communicating. The UI should display HTTP Response Code = 200 for the respective plugin.

Related Information

Installing the Ranger Key Management Service

Using the Ranger Key Management Service

Ranger KMS can be accessed at the Ranger admin URL, http://$hostname:6080. Note, however, that the login user for Ranger KMS is different than that for Ranger. Logging on as the Ranger KMS admin user leads to a different set of screens.

Role Separation

By default, Ranger admin uses a different admin user (keyadmin) to manage access policies and keys for Ranger KMS.

The person accessing Ranger KMS via the keyadmin user should be a different person than the administrator who works with regular Ranger access policies. This approach separates encryption work (encryption keys and policies) from Hadoop cluster management and access policy management.

Accessing the Ranger KMS Web UI

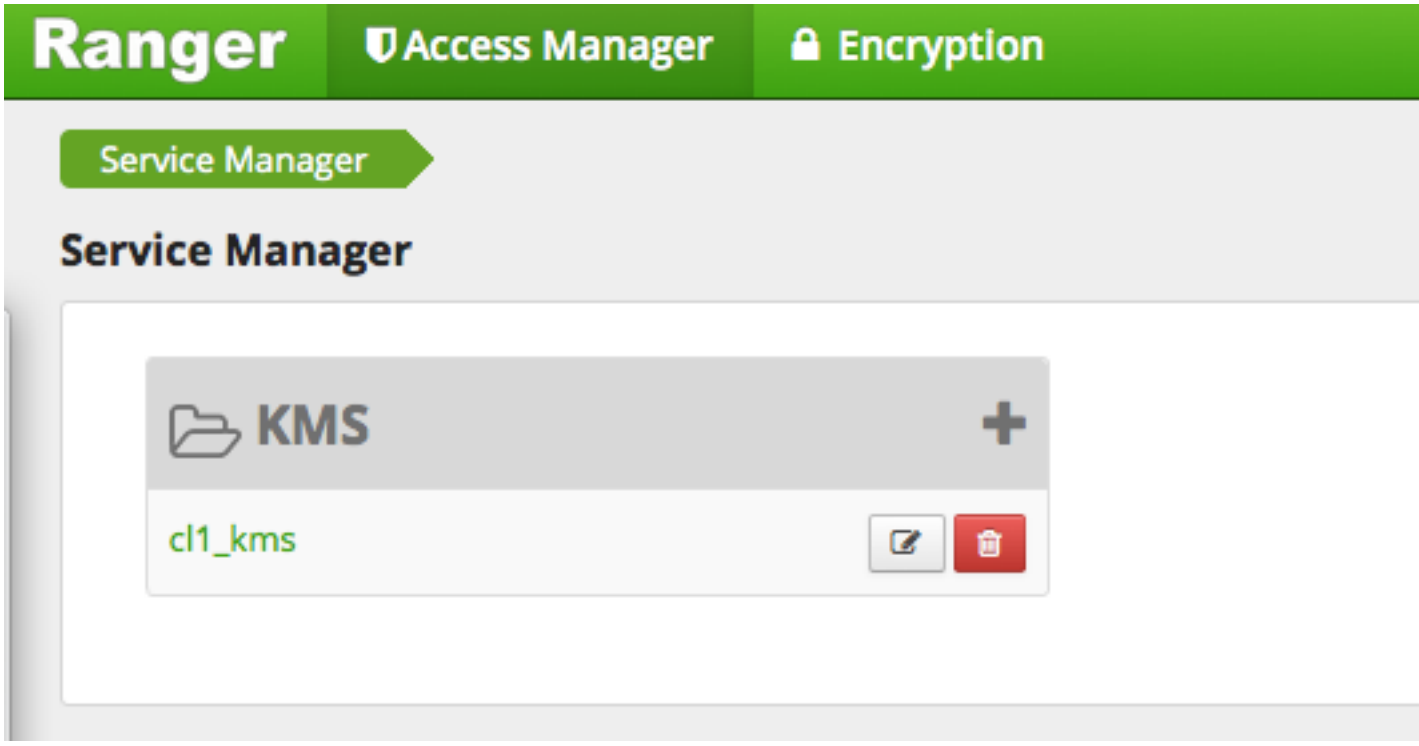
How and where to access the Ranger KMS Web UI.

To access Ranger KMS, log in as user keyadmin, password keyadmin.

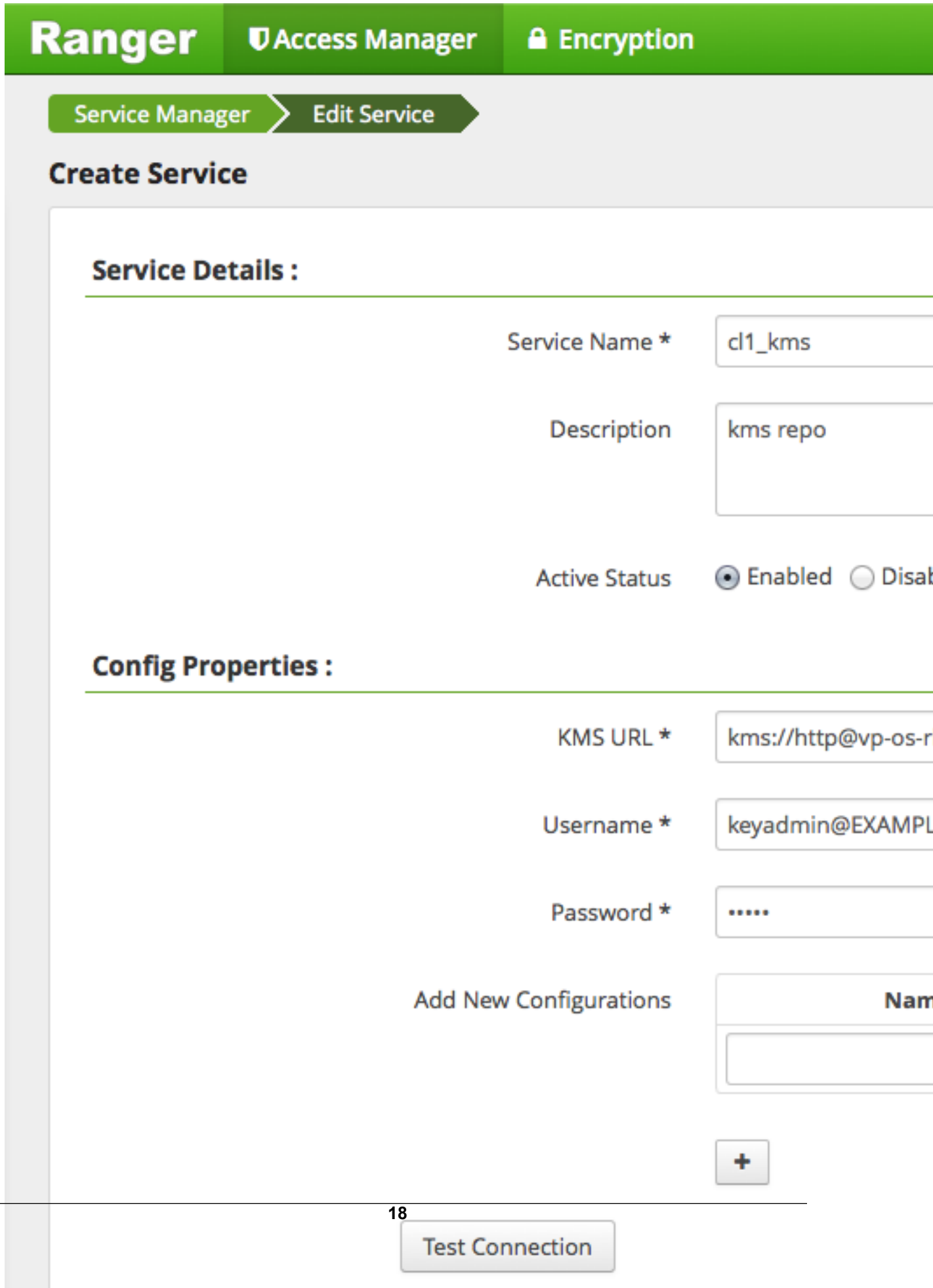
Note:

Change the password after you log in.

After logging in, you will see the Service Manager screen. To view or edit Ranger KMS repository properties, click on the edit button next to the repository name:



You will see a list of service details and config properties for the repository:


Ranger
Access Manager
Encryption
Service Manager
Edit Service
Create Service
Service Details :
Service Name *
cl1_kms
Description
kms repo
Active Status
Enabled
Disab
Config Properties :
KMS URL *
kms://http@vp-os-r
Username *
keyadmin@EXAMPL
Password *
•••••
Add New Configurations
Nam
+
Test Connection

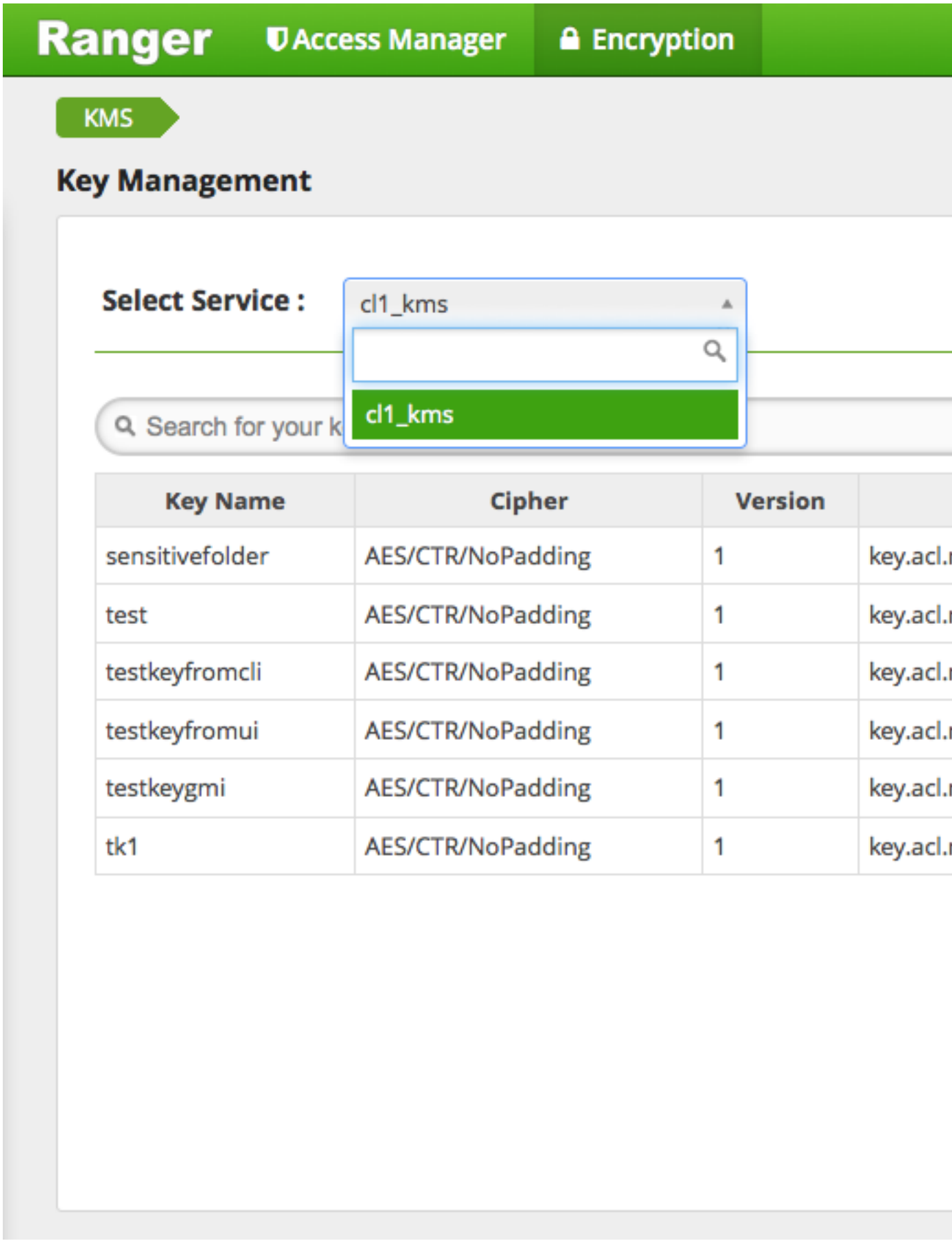
List and Create Keys

How to list and create keys, when using the Ranger KMS.

Procedure

To list existing keys:

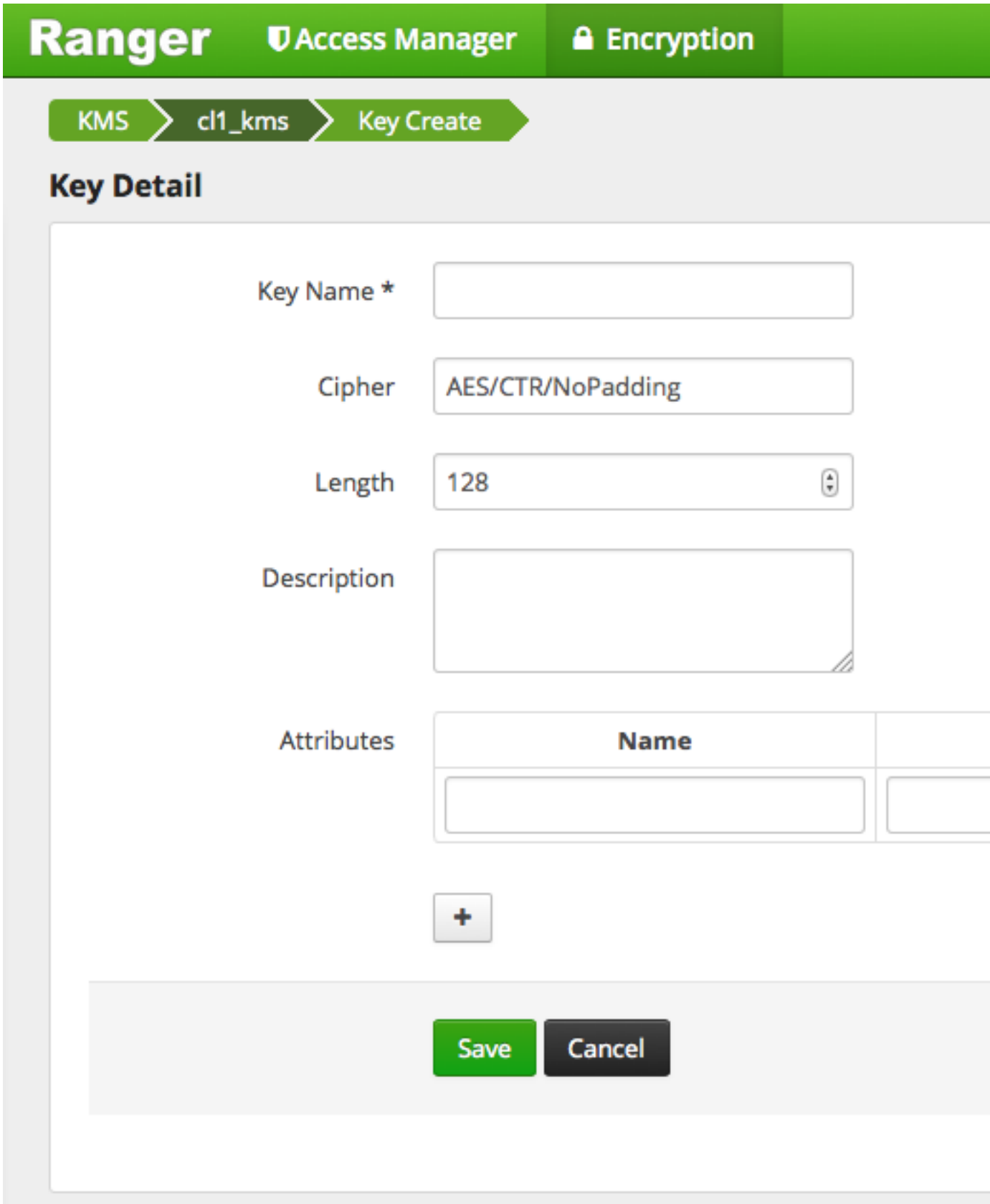
1. Log in to Ranger as user keyadmin, password $keyadmin.
2. Choose the Encryption tab at the top of the Ranger Web UI screen.
3. Select the Ranger KMS service from the drop-down list.


Ranger
Access Manager
Encryption
KMS
Key Management
Select Service :
cl1_kms
cl1_kms
Search for your k
Key Name
Cipher
Version
sensitivefolder
AES/CTR/NoPadding
1
test
AES/CTR/NoPadding
1
testkeyfromcli
AES/CTR/NoPadding
1
testkeyfromui
AES/CTR/NoPadding
1
testkeygmi
AES/CTR/NoPadding
1
tk1
AES/CTR/NoPadding
1

The existing keys are displayed.

To create a new key:

4. Click on "Add New Key".
5. Add a valid key name.
6. Select the cipher name. Ranger supports AES/CTR/NoPadding as the cipher suite.
7. Specify the key length, 128 or 256 bits.
8. Add other attributes as needed, and click **Save**.



Roll Over an Existing Key

How to roll over an existing key, when using the Ranger KMS.

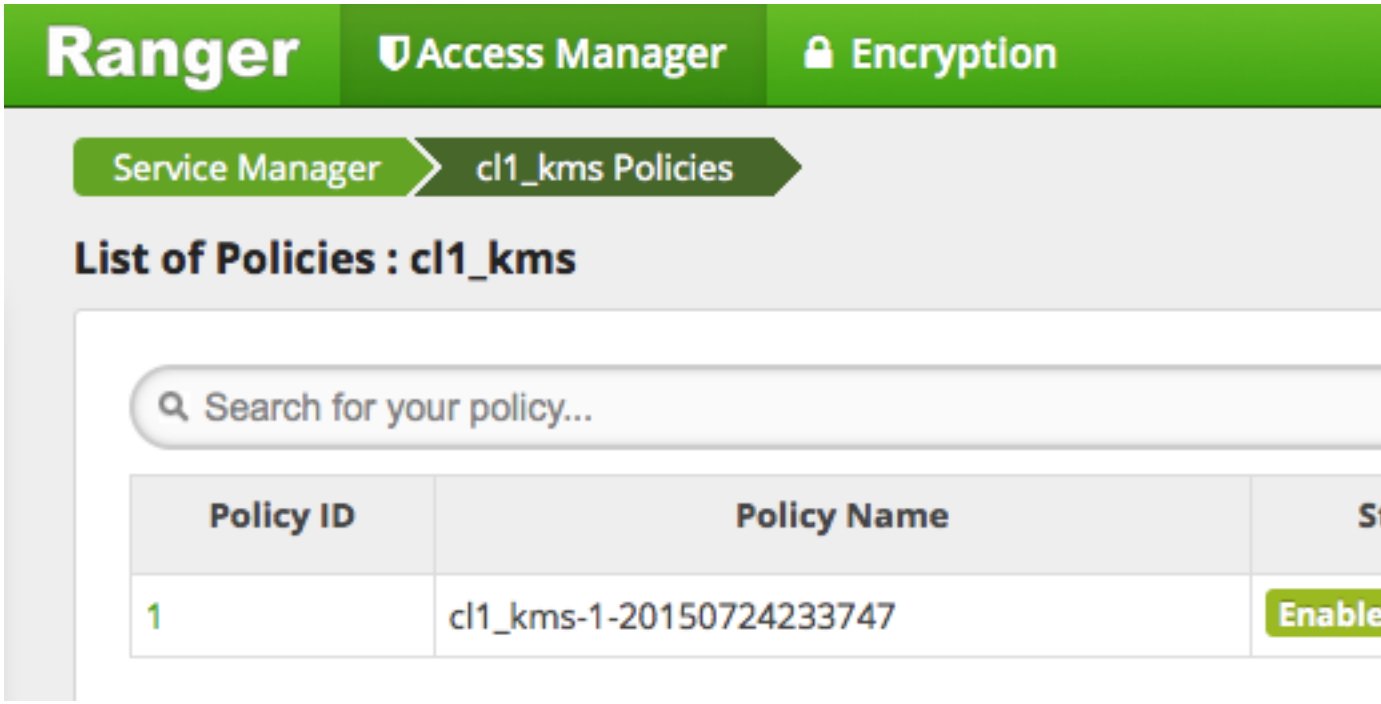
About this task

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.

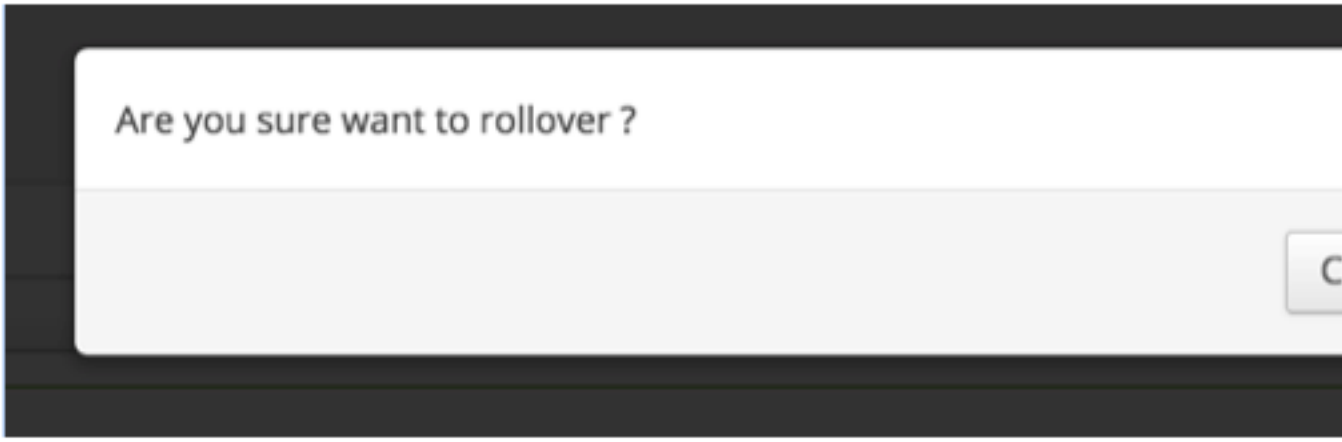
After a key is rotated in Ranger KMS, new files will have the file key encrypted by the new master key for the encryption zone.

Procedure

1. Log in to Ranger as user keyadmin, password $keyadmin.
2. To rotate a key, click the edit button next to the key name in the list of keys:



3. Edit the key information, and then press Save.
4. When asked to confirm the rollover, click "OK":



Delete a Key

How to delete a key, when using the Ranger KMS.

About this task

Attention:

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

Procedure

1. Log in to Ranger as user keyadmin, password $keyadmin.
2. Choose the Encryption tab at the top of the Ranger Web UI screen.
3. Select Ranger KMS service from the drop-down list.
4. Click on the delete symbol next to the key.
5. You will see a confirmation pop-up window; confirm or cancel.

Ranger KMS Properties

This topic describes configuration properties for the Ranger Key Management Service (KMS).

Table 1: Properties in Advanced dbks-site Menu (dbks-site.xml)

Property Name	Default Value	Description
ranger.ks.masterkey.credential.alias	ranger.ks.masterkey.password	Credential alias used for masterkey.
ranger.ks.jpa.jdbc.user	rangerkms	Database username used for operation.
ranger.ks.jpa.jdbc.url	jdbc:log4jdbc:mysql://localhost:3306/rangerkms	JDBC connection URL for database.
ranger.ks.jpa.jdbc.password	_ (default it's encrypted)	Database user's password.
ranger.ks.jpa.jdbc.driver	net.sf.log4jdbc.DriverSpy	Driver used for database.
ranger.ks.jpa.jdbc.dialect	org.eclipse.persistence.platform.database.MySQLPlatform	Dialect used for database.
ranger.ks.jpa.jdbc.credential. provider.path	/etc/ranger/kms/rangerkms.jceks	Credential provider path.
ranger.ks.jpa.jdbc.credential.alias	ranger.ks.jdbc.password	Credential alias used for password.
ranger.ks.jdbc.sqlconnectorjar	/usr/share/java/mysql-connector-java.jar	Driver jar used for database.
ranger.db.encrypt.key.password	_ (Default; it's encrypted)	Password used for encrypting the Master Key.
hadoop.kms.blacklist.DECRYPT_EEK	hdfs	Blacklist for decrypt EncryptedKey CryptoExtension operations. This can have multiple user IDs in a comma separated list. e.g. stormuser,yarn,hdfs.

Table 2: Properties in Advanced kms-env

Property Name	Default Value	Description
Kms User	kms	Ranger KMS process will be started using this user.
Kms Group	kms	Ranger KMS process will be started using this group.
LD library path		LD library path (basically used when the db flavor is SQLA). Example: /opt/sqlanywhere17/lib64
kms_port	9292	Port used by Ranger KMS.

Property Name	Default Value	Description
kms_log_dir	/var/log/ranger/kms	Directory where the Ranger KMS log will be generated.

Table 3: Properties in Advanced kms-properties (install.properties)

Property Name	Default Value	Description
db_user	rangerkms	Database username used for the operation.
db_root_user		Database root username. Default is blank. Specify the root user.
db_root_password		Database root user's password. Default is blank. Specify the root user password.
db_password		Database user's password for the operation. Default is blank. Specify the Ranger KMS database password.
db_name	rangerkms	Database name for Ranger KMS.
db_host	<FQDN of instance where the Ranger KMS is installed>	Hostname where the database is installed. Note: Check the hostname for DB and change it accordingly.
SQL_CONNECTOR_JAR	/usr/share/java/mysql-connector.jar	Location of DB client library.
REPOSITORY_CONFIG_USERNAME	keyadmin	User used in default repo for Ranger KMS.
REPOSITORY_CONFIG_PASSWORD	keyadmin	Password for user used in default repo for Ranger KMS.
KMS_MASTER_KEY_PASSWD		Password used for encrypting the Master Key. Default value is blank. Set the master key to any string.
DB_FLAVOR	MYSQL	Database flavor used for Ranger KMS. Supported values: MYSQL, SQLA, ORACLE, POSTGRES, MSSQL

Table 4: Properties in Advanced kms-site (kms-site.xml)

Property Name	Default Value	Description
hadoop.security.keystore. JavaKeyStoreProvider.password	none	If using the JavaKeyStoreProvide, the password for the keystore file.
hadoop.kms.security. authorization.manager	org.apache.ranger. authorization.kms. authorizer.RangerKmsAuthorizer	Ranger KMS security authorizer.
hadoop.kms.key.provider.uri	dbks://http@localhost:9292/kms	URI of the backing KeyProvider for the KMS.
hadoop.kms.current.key. cache.timeout.ms	30000	Expiry time for the KMS current key cache, in milliseconds. This affects getCurrentKey operations.
hadoop.kms.cache.timeout.ms	600000	Expiry time for the KMS key version and key metadata cache, in milliseconds. This affects getKeyVersion and getMetadata.

Property Name	Default Value	Description
hadoop.kms.cache.enable	true	Whether the KMS will act as a cache for the backing KeyProvider. When the cache is enabled, operations like getKeyVersion, getMetadata, and getCurrentKey will sometimes return cached data without consulting the backing KeyProvider. Cached values are flushed when keys are deleted or modified. Note: This setting is beneficial if Single KMS and single mode are used. If this is set to true when multiple KMSs are used, or when the key operations are from different modes (Ranger UI, CURL, or hadoop command), it might cause inconsistency.
hadoop.kms.authentication.type	simple	Authentication type for the Ranger KMS. Can be either “simple” or “kerberos”.
hadoop.kms.authentication.signer.secret.provider.zookeeper.path	/hadoop-kms/hadoop-auth-signature-secret	The ZooKeeper ZNode path where the Ranger KMS instances will store and retrieve the secret from.
hadoop.kms.authentication.signer.secret.provider.zookeeper.kerberos.principal	kms/#HOSTNAME#	The Kerberos service principal used to connect to ZooKeeper
hadoop.kms.authentication.signer.secret.provider.zookeeper.kerberos.keytab	/etc/hadoop/conf/kms.keytab	The absolute path for the Kerberos keytab with the credentials to connect to ZooKeeper.
hadoop.kms.authentication.signer.secret.provider.zookeeper.connection.string	#HOSTNAME#:#PORT#,...	The ZooKeeper connection string, a list of hostnames and port comma separated. For example: <FQDN for first instance>:2181,<FQDN for second instance>:2181
hadoop.kms.authentication.signer.secret.provider.zookeeper.auth.type	kerberos	ZooKeeper authentication type: 'none' or 'sasl' (Kerberos)
hadoop.kms.authentication. signer.secret.provider	random	Indicates how the secret to sign authentication cookies will be stored. Options are 'random' (default), 'string', and zookeeper'. If you have multiple Ranger KMS instances, specify 'zookeeper'.
hadoop.kms.authentication.kerberos.principal	HTTP/localhost	The Kerberos principal to use for the HTTP endpoint. The principal must start with 'HTTP/' as per the Kerberos HTTP SPNEGO specification.
hadoop.kms.authentication.kerberos.name.rules	DEFAULT	Rules used to resolve Kerberos principal names.
hadoop.kms.authentication.kerberos.keytab	${user.home}/kms.keytab	Path to the keytab with credentials for the configured Kerberos principal.
hadoop.kms.audit.aggregation.window.ms	10000	Specified in ms. Duplicate audit log events within this aggregation window are quashed to reduce log traffic. A single message for aggregated events is printed at the end of the window, along with a count of the number of aggregated events.

Table 5: Properties in Advanced ranger-kms-audit (ranger-kms-audit.xml)

Property Name	Default Value	Description
Audit provider summary enabled		Enable audit provider summary.
xasecure.audit.is.enabled	true	Enable audit.
xasecure.audit.destination. solr.zookeepers	none	Specify solr zookeeper string.
xasecure.audit.destination.solr.urls	{{ranger_audit_solr_urls}}	Specify solr URL. Note: In Ambari this value is populated from the Ranger Admin by default.
xasecure.audit.destination. solr.batch.filespool.dir	/var/log/ranger/kms/audit/solr/spool	Directory for solr audit spool.
Audit to SOLR		Enable audit to solr.
xasecure.audit.destination.hdfs.dir	hdfs://NAMENODE_HOST:8020/ranger/audit	HDFS directory to write audit. Note: Make sure the service user has required permissions.
xasecure.audit.destination. hdfs.batch.filespool.dir	/var/log/ranger/kms/audit/hdfs/spool	Directory for HDFS audit spool.
Audit to HDFS		Enable hdfs audit.
xasecure.audit.destination.db.user	{{xa_audit_db_user}}	xa audit db user Note: In Ambari this value is populated from the Ranger Admin by default.
xasecure.audit.destination. db.password	encrypted (it's in encrypted format)	xa audit db user password Note: In Ambari this value is populated from the Ranger Admin by default.
xasecure.audit.destination.db.jdbc.url	{{audit_jdbc_url}}	Database JDBC URL for xa audit. Note: In Ambari the value for this is populated from the Ranger Admin by default.
xasecure.audit.destination. db.jdbc.driver	{{jdbc_driver}}	Database JDBC driver. Note: In Ambari this value is populated from the Ranger Admin by default.
xasecure.audit.destination. db.batch.filespool.dir	/var/log/ranger/kms/audit/db/spool	Directory for database audit spool.
Audit to DB		Enable audit to database.
xasecure.audit.credential.provider.file	jceks://file{{credential_file}}	Credential provider file.

Table 6: Properties in Advanced ranger-kms-policymgr-ssl

Property Name	Default Value	Description
xasecure.policymgr.clientssl. truststore.password	changeit	Password for the truststore.
xasecure.policymgr.clientssl. truststore	/usr/hdp/current/ranger-kms/conf/ranger-plugin-truststore.jks	jks file for truststore
xasecure.policymgr.clientssl. keystore.password	myKeyFilePassword	Password for keystore.
xasecure.policymgr.clientssl. keystore.credential.file	jceks://file{{credential_file}}	Java keystore credential file.

Property Name	Default Value	Description
xasecure.policymgr.clientssl. keystore	/usr/hdp/current/ranger-kms/conf/ranger-plugin-keystore.jks	Java keystore file.
xasecure.policymgr.clientssl. truststore.credential.file	jceks://file{{credential_file}}	Java truststore file.

Table 7: Properties in Advanced ranger-kms-security

Property Name	Default Value	Description
ranger.plugin.kms.service.name	<default name for Ranger KMS Repo>	Name of the Ranger service containing policies for the KMS instance. Note: In Ambari the default value is <clusterName>_kms.
ranger.plugin.kms.policy.source.impl	org.apache.ranger.admin.client. RangerAdminRESTClient	Class to reterive policies from the source.
ranger.plugin.kms.policy.rest.url	{{policymgr_mgr_url}}	URL for Ranger Admin.
ranger.plugin.kms.policy.rest. ssl.config.file	/etc/ranger/kms/conf/ranger-policymgr-ssl.xml	Path to the file containing SSL details for contacting the Ranger Admin.
ranger.plugin.kms.policy. pollIntervalMs	30000	Time interval to poll for changes in policies.
ranger.plugin.kms.policy.cache.dir	/etc/ranger/{{repo_name}}/policycache	Directory where Ranger policies are cached after successful retrieval from the source.

Troubleshooting Ranger KMS

Troubleshooting suggestions for Ranger KMS.

Table 8: Troubleshooting Suggestions

Issue	Action
Not able to install Ranger KMS	Check to see if ranger admin is running, verify DB.
Not able to start Ranger KMS	Check the Ranger KMS log. If there is a message about illegal key size, make sure unlimited strength JCE is available.
Hadoop key commands fail	Make sure Ranger KMS client properties are updated in hdfs config.
Not able to create keys from Ranger UI	Make sure that the keyadmin user (or any custom user) configured in the KMS repository is added to proxy properties in the custom kms-site.xml file.

HDFS "Data at Rest" Encryption

Encryption is a form of data security that is required in industries such as healthcare and the payment card industry. Hadoop provides several ways to encrypt stored data.

- The lowest level of encryption is volume encryption, which protects data after physical theft or accidental loss of a disk volume. The entire volume is encrypted; this approach does not support finer-grained encryption of specific files or directories. In addition, volume encryption does not protect against viruses or other attacks that occur while a system is running.

- Application level encryption (encryption within an application running on top of Hadoop) supports a higher level of granularity and prevents "rogue admin" access, but adds a layer of complexity to the application architecture.
- A third approach, HDFS data at rest encryption, encrypts selected files and directories stored ("at rest") in HDFS. This approach uses specially designated HDFS directories known as "encryption zones."

This chapter focuses on the third approach, HDFS data at rest encryption. The chapter is intended as an introductory quick start to HDFS data at rest encryption. Content will be updated regularly.

HDFS Encryption Overview

HDFS data at rest encryption implements end-to-end encryption of data read from and written to HDFS. End-to-end encryption means that data is encrypted and decrypted only by the client. HDFS does not have access to unencrypted data or keys.

HDFS encryption involves several elements:

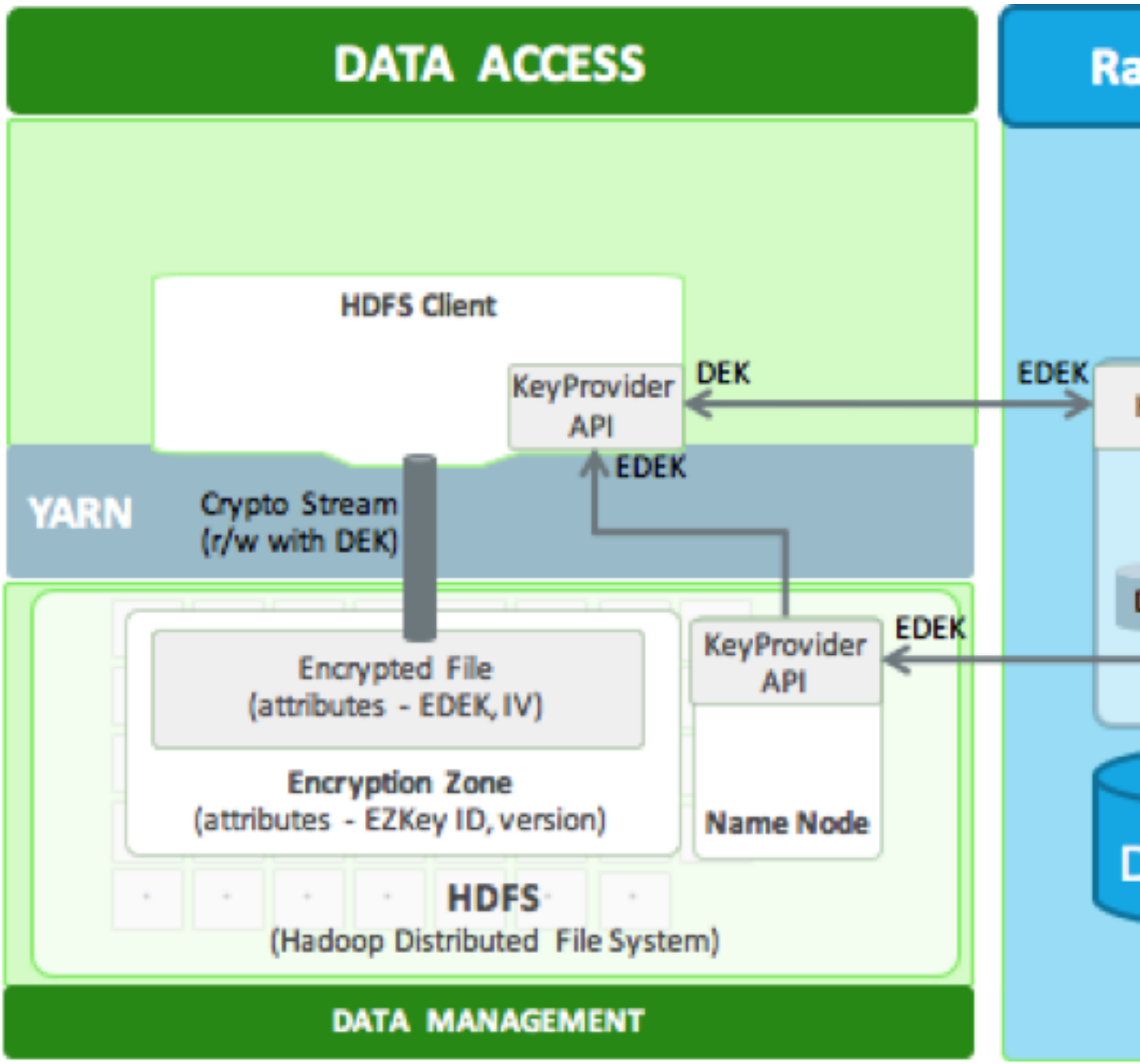
- Encryption key: A new level of permission-based access protection, in addition to standard HDFS permissions.
- HDFS encryption zone: A special HDFS directory within which all data is encrypted upon write, and decrypted upon read.
  - Each encryption zone is associated with an encryption key that is specified when the zone is created.
  - Each file within an encryption zone has a unique encryption key, called the "data encryption key" (DEK).
  - HDFS does not have access to DEKs. HDFS DataNodes only see a stream of encrypted bytes. HDFS stores "encrypted data encryption keys" (EDEKs) as part of the file's metadata on the NameNode.
  - Clients decrypt an EDEK and use the associated DEK to encrypt and decrypt data during write and read operations.
- Ranger Key Management Service (Ranger KMS): An open source key management service based on Hadoop’s KeyProvider API.

  For HDFS encryption, the Ranger KMS has three basic responsibilities:

  - Provide access to stored encryption zone keys.
  - Generate and manage encryption zone keys, and create encrypted data keys to be stored in Hadoop.
  - Audit all access events in Ranger KMS.

  Note : This chapter is intended for security administrators who are interested in configuring and using HDFS encryption. For more information about Ranger KMS, see “Ranger KMS Administration”.

HDFS Encryption Components



Role Separation

Access to the key encryption/decryption process is typically restricted to end users. This means that encrypted keys can be safely stored and handled by HDFS, because the HDFS admin user does not have access to them.

This role separation requires two types of HDFS administrator accounts:

- HDFS service user: the system-level account associated with HDFS (hdfs by default).
- HDFS admin user: an account in the hdfs supergroup, which is used by HDFS administrators to configure and manage HDFS.

> **Note:** For clear segregation of duties, we recommend that you restrict use of the hdfs account to system/interprocess use. Do not provide its password to physical users. A (human) user who administers HDFS should only access HDFS through an admin user account created specifically

for that purpose. For more information about creating an HDFS admin user, see “Create an HDFS Admin User”.

Other services may require a separate admin account for clusters with HDFS encryption zones. For service-specific information, see “Configuring HDP Services for HDFS Encryption”.

Related Information

Configuring HDP Services for HDFS Encryption

Ranger KMS Administration

Create an HDFS Admin User

Configuring and Starting the Ranger Key Management Service (Ranger KMS)

In a typical environment, a security administrator will set up the Ranger Key Management Service. For information about installing and configuring the Ranger KMS, see “Ranger KMS Administration”.

Related Information

Ranger KMS Administration

Configuring and Using HDFS "Data at Rest" Encryption

After the Ranger KMS has been set up and the NameNode and HDFS clients have been configured, an HDFS administrator can use the hadoop key and hdfs crypto command-line tools to create encryption keys and set up new encryption zones.

Note: You should create a separate HDFS Admin user account for HDFS Data at Rest Encryption.

The overall workflow is as follows:

1. Create an HDFS encryption zone key that will be used to encrypt the file-level data encryption key for every file in the encryption zone. This key is stored and managed by Ranger KMS.
2. Create a new HDFS folder. Specify required permissions, owner, and group for the folder.
3. Using the new encryption zone key, designate the folder as an encryption zone.
4. Configure client access. The user associated with the client application needs sufficient permission to access encrypted data. In an encryption zone, the user needs file/directory access (through Posix permissions or Ranger access control), as well as access for certain key operations. To set up ACLs for key-related operations, see “Ranger KMS Administration”.

After permissions are set, Java API clients and HDFS applications with sufficient HDFS and Ranger KMS access privileges can write and read to/from files in the encryption zone.

Related Information

Ranger KMS Administration

Preparing the Environment

HDP supports hardware acceleration with Advanced Encryption Standard New Instructions (AES-NI). Compared with the software implementation of AES, hardware acceleration offers an order of magnitude faster encryption/decryption.

CPU Support for AES NI optimization

To use AES-NI optimization you need CPU and library support, described in the following subsections.

AES-NI optimization requires an extended CPU instruction set for AES hardware acceleration.

There are several ways to check for this; for example:

```
$ cat /proc/cpuinfo | grep aes
```

Look for output with flags and 'aes'.

Library Support for AES NI optimization

You will need a version of the libcrypto.so library that supports hardware acceleration, such as OpenSSL 1.0.1e. (Many OS versions have an older version of the library that does not support AES-NI.)

A version of the libcrypto.so library with AES-NI support must be installed on HDFS cluster nodes and MapReduce client hosts -- that is, any host from which you issue HDFS or MapReduce requests. The following instructions describe how to install and configure the libcrypto.so library.

RHEL/CentOS 6.5 or later

On HDP cluster nodes, the installed version of libcrypto.so supports AES-NI, but you will need to make sure that the symbolic link exists:

$ sudo ln -s /usr/lib64/libcrypto.so.1.0.1e /usr/lib64/libcrypto.so

On MapReduce client hosts, install the openssl-devel package:

$ sudo yum install openssl-devel

Verifying AES NI Support

To verify that a client host is ready to use the AES-NI instruction set optimization for HDFS encryption, use the following command:

hadoop checknative

You should see a response similar to the following:

```
15/08/12 13:48:39 INFO bzip2.Bzip2Factory: Successfully loaded & initialized
 native-bzip2 library system-native
14/12/12 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 correct version.

Create an Encryption Key

Create a "master" encryption key for the new encryption zone. Each key will be specific to an encryption zone. You can create a new encryption key via Ranger KMS (recommended) or the CLI.

About this task

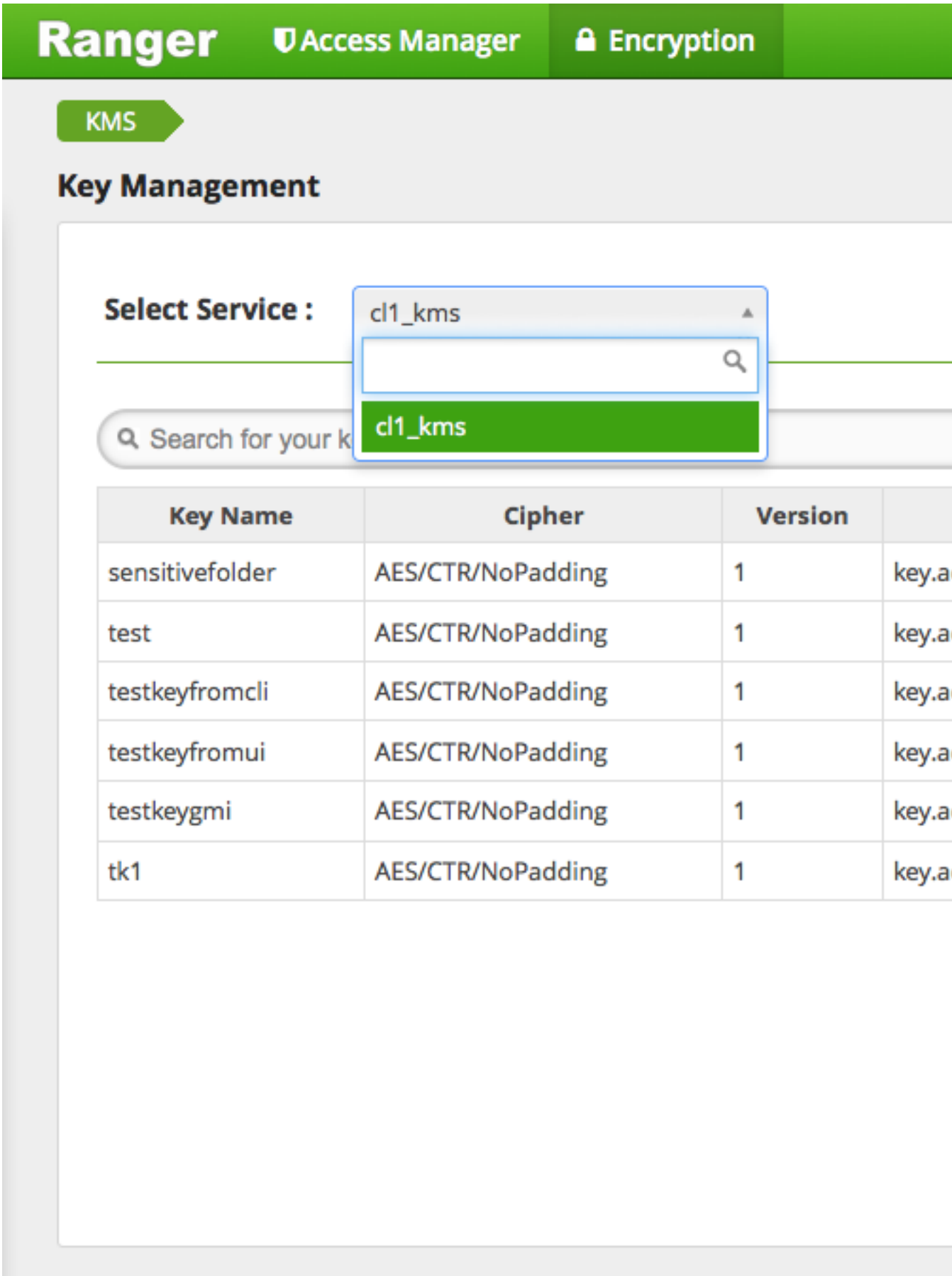
Ranger supports AES/CTR/NoPadding as the cipher suite. (The associated property is listed under HDFS -> Configs in the Advanced hdfs-site list.)

Key size can be 128 or 256 bits.

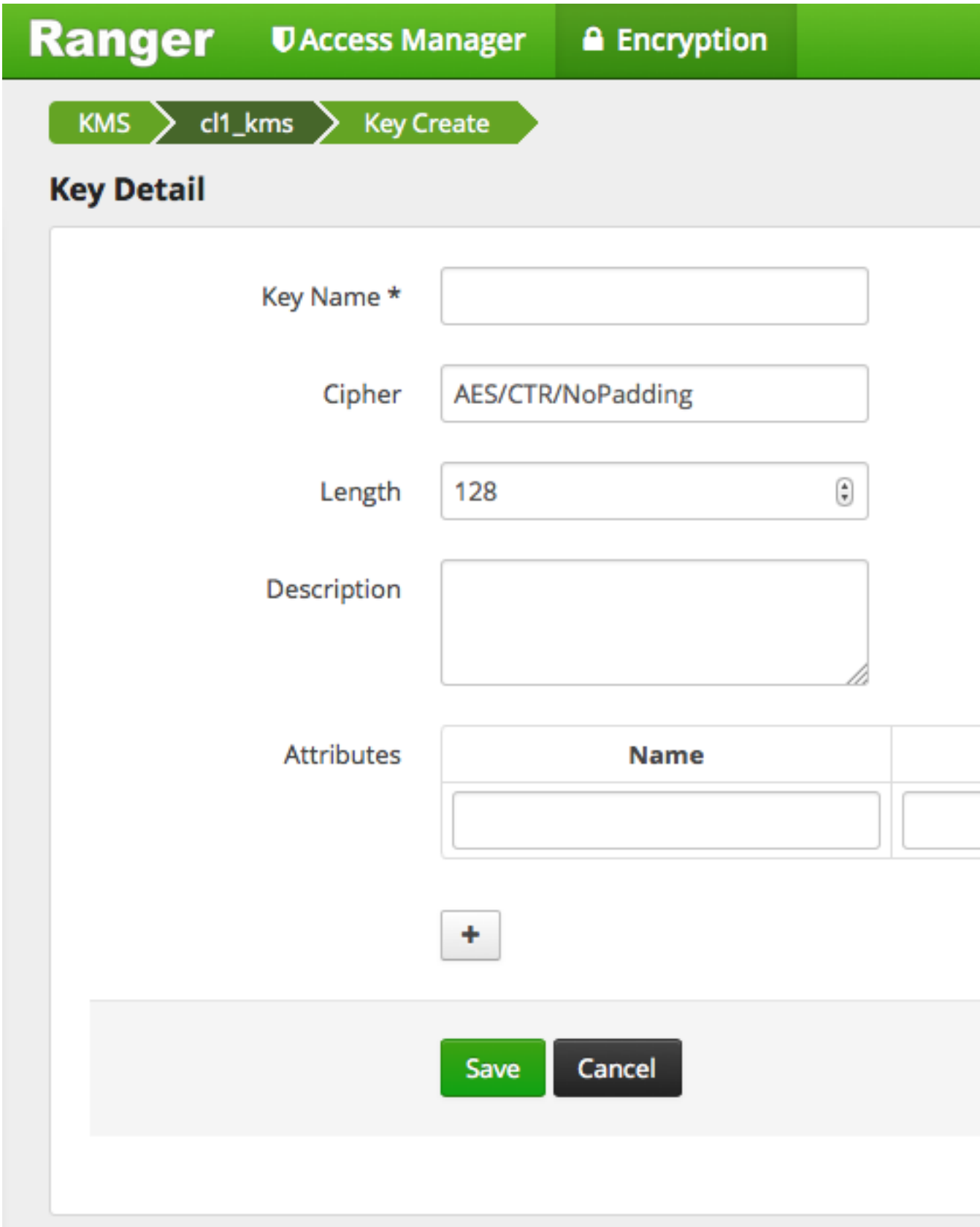
Recommendation: create a new superuser for key management. In the following examples, superuser encr creates the key. This separates the data access role from the encryption role, strengthening security.

Procedure

- To create an Encryption Key using Ranger KMS (Recommended):
  a) Log in to Ranger as user keyadmin, password $keyadmin.
  b) In the Ranger Web UI screen, choose the Encryption tab at the top of the screen.
  c) Select the KMS service from the drop-down list.


Ranger
Access Manager
Encryption
KMS
Key Management
Select Service :
cl1_kms
cl1_kms
Search for your k
Key Name
Cipher
Version
sensitivefolder
AES/CTR/NoPadding
1
test
AES/CTR/NoPadding
1
testkeyfromcli
AES/CTR/NoPadding
1
testkeyfromui
AES/CTR/NoPadding
1
testkeygmi
AES/CTR/NoPadding
1
tk1
AES/CTR/NoPadding
1

d) Click on "Add New Key":
e) Add a valid key name.
f) Select the cipher name. Ranger supports AES/CTR/NoPadding as the cipher suite.
g) Specify the key length, 128 or 256 bits.
h) Add other attributes as needed, and then save the key.



- To create an Encryption Key using the CLI:

a) The full syntax of the hadoop key create command is as follows:

```
[create <keyname> [-cipher <cipher>]
[-size <size>]
[-description <description>]
[-attr <attribute=value>]
[-provider <provider>]
[-help]]
```

```
# su - encr
# hadoop key create <key_name> [-size <number-of-bits>]
```

The default key size is 128 bits. The optional -size parameter supports 256-bit keys, and requires the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy File on all hosts in the cluster. For installation information, see “Installing the JCE”.

```
# su - encr
# hadoop key create key1
```

To verify creation of the key, list the metadata associated with the current user: # hadoop key list -metadata

Related Information

Using the Ranger Key Management Service

Install the JCE for Kerberos

Create an Encryption Zone

How to create an encryption zone when configuring HDFS encryption.

About this task

Each encryption zone must be defined using an empty directory and an existing encryption key. An encryption zone cannot be created on top of a directory that already contains data.

Recommendation: use one unique key for each encryption zone.

Use the crypto createZone command to create a new encryption zone. The syntax is:

-createZone -keyName <keyName> -path <path>

where:

- -keyName: specifies the name of the key to use for the encryption zone.
- -path specifies the path of the encryption zone to be created. It must be an empty directory.

Procedure

1. As HDFS administrator, create a new empty directory.
   # hdfs dfs -mkdir /zone_encr
2. Using the encryption key, make the directory an encryption zone.
   # hdfs crypto -createZone -keyName key1 -path /zone_encr
   When finished, the NameNode will recognize the folder as an HDFS encryption zone.
3. To verify creation of the new encryption zone, run the crypto -listZones command as an HDFS administrator: -listZones.

   Note:

   The following property (in the hdfs-default.xml file) causes listZone requests to be batched. This improves NameNode performance. The property specifies the maximum number of zones that will be returned in a batch.

dfs.namenode.list.encryption.zones.num.responses

The default is 100.

You should see the encryption zone and its key. For example:

```
$ hdfs crypto -listZones
/zone-encr  key1
```

What to do next

(Optional) To remove an encryption zone, delete the root directory of the zone. For example: hdfs dfs -rm -R /zone_encr.

Copying Files to or from an Encryption Zone

Information on how to copy existing files to or from an encryption zone, use a tool like distcp.

Note: for separation of administrative roles, do not use the hdfs user to create encryption zones. Instead, designate another administrative account for creating encryption keys and zones. See “Appendix: Creating an HDFS Admin User” for more information.

The files will be encrypted using a file-level key generated by the Ranger Key Management Service.

DistCp Considerations

DistCp is commonly used to replicate data between clusters for backup and disaster recovery purposes. This operation is typically performed by the cluster administrator, via an HDFS superuser account.

To retain this workflow when using HDFS encryption, a new virtual path prefix has been introduced, /.reserved/raw/. This virtual path gives super users direct access to the underlying encrypted block data in the file system, allowing super users to distcp data without requiring access to encryption keys. This also avoids the overhead of decrypting and re-encrypting data. The source and destination data will be byte-for-byte identical, which would not be true if the data were re-encrypted with a new EDEK.

Attention:

When using /.reserved/raw/ to distcp encrypted data, make sure you preserve extended attributes with the -px flag. This is necessary because encrypted attributes such as the EDEK are exposed through extended attributes; they must be preserved to be able to decrypt the file. For example:

sudo -u encr hadoop distcp -px hdfs:/cluster1-namenode:50070/.reserved/raw/apps/enczone hdfs:/cluster2-namenode:50070/.reserved/raw/apps/enczone

This means that if the distcp operation is initiated at or above the encryption zone root, it will automatically create a new encryption zone at the destination (if one does not already exist).

Recommendation: To avoid potential mishaps, first create identical encryption zones on the destination cluster.

Copying between encrypted and unencrypted locations

By default, distcp compares file system checksums to verify that data was successfully copied to the destination.

When copying between an unencrypted and encrypted location, file system checksums will not match because the underlying block data is different. In this case, specify the -skipcrccheck and -update flags to avoid verifying checksums.

Related Information

Create an HDFS Admin User

Reading and Writing Files from or to an Encryption Zone

Clients and HDFS applications with sufficient HDFS and Ranger KMS permissions can read and write files from/to an encryption zone.

Overview of the client write process

1. The client writes to the encryption zone.
2. The NameNode checks to make sure that the client has sufficient write access permissions. If so, the NameNode asks Ranger KMS to create a file-level key, encrypted with the encryption zone master key.
3. The Namenode stores the file-level encrypted data encryption key (EDEK) generated by Ranger KMS as part of the file's metadata, and returns the EDEK to the client.
4. The client asks Ranger KMS to decode the EDEK (to DEK), and uses the DEK to write encrypted data. Ranger KMS checks for permissions for the user before decrypting EDEK and producing the DEK for the client.

Overview of the client read process

1. The client issues a read request for a file in an encryption zone.
2. The NameNode checks to make sure that the client has sufficient read access permissions. If so, the NameNode returns the file's EDEK and the encryption zone key version that was used to encrypt the EDEK.
3. The client asks Ranger KMS to decrypt the EDEK. Ranger KMS checks for permissions to decrypt EDEK for the end user.
4. Ranger KMS decrypts and returns the (unencrypted) data encryption key (DEK).
5. The client uses the DEK to decrypt and read the file.

The preceding steps take place through internal interactions between the DFSClient, the NameNode, and Ranger KMS.

Examples

In the following example, the /zone_encr directory is an encrypted zone in HDFS.

To verify this, use the crypto -listZones command (as an HDFS administrator). This command lists the root path and the zone key for the encryption zone. For example:

```
# hdfs crypto -listZones
/zone_encr  key1
```

Additionally, the /zone_encr directory has been set up for read/write access by the hive user:

```
# hdfs dfs -ls /
 …
drwxr-x---   - hive   hive            0 2015-01-11 23:12 /zone_encr
```

The hive user can, therefore, write data to the directory.

The following examples use the copyFromLocal command to move a local file into HDFS.

```
[hive@blue ~]# hdfs dfs -copyFromLocal web.log /zone_encr
[hive@blue ~]# hdfs dfs -ls /zone_encr
Found 1 items
-rw-r--r--   1 hive hive       1310 2015-01-11 23:28 /zone_encr/web.log
```

The hive user can read data from the directory, and can verify that the file loaded into HDFS is readable in its unencrypted form.

```
[hive@blue ~]# hdfs dfs -copyToLocal /zone_encr/web.log read.log
```

```
[hive@blue ~]# diff web.log read.log
```

Users without access to KMS keys will be able to see file names (via the -ls command), but they will not be able to write data or read from the encrypted zone. For example, the hdfs user lacks sufficient permissions, and cannot access the data in /zone_encr:

```
[hdfs@blue ~]# hdfs dfs -copyFromLocal install.log /zone_encr
copyFromLocal: Permission denied: user=hdfs, access=EXECUTE, inode="/
zone_encr":hive:hive:drwxr-x---

[hdfs@blue ~]# hdfs dfs -copyToLocal /zone_encr/web.log read.log
copyToLocal: Permission denied: user=hdfs, access=EXECUTE, inode="/
zone_encr":hive:hive:drwxr-x---
```

Deleting Files from an Encryption Zone with Trash Enabled

Information on deleting files from an encryption zone with trash enabled.

The trash location for encrypted HDFS files is different than the default trash location for unencrypted files (/user/$USER/.Trash/Current/OriginalPathToDeletedFile).

When trash is enabled and an encrypted file is deleted, the file is moved to the .Trash subdirectory under the root of the encryption zone as /EncryptionZoneRoot/.Trash/$USER/Current/OriginalPathToDeletedFile. The file remains encrypted without additional decryption/re-encryption overhead during the move to trash. The move operation preserves the name of the user who executes the deletion, and the full path of the deleted file.

Example

For example, if user hdp-admin deletes file /zone_name/file1 using the following command:

hdfs dfs -rm /zone_name/file1

file1 will remain encrypted, and it will be moved to the following location within the encryption zone:

/zone_name/.Trash/hdp-admin/Current/zone_name/file1

A trash checkpoint will be created for the .Trash subdirectory in each encryption zone. Checkpoints will be deleted/created according to the value of fs.trash.checkpoint.interval (number of minutes between trash checkpoints). A checkpoint for this example would be:

/zone_name/.Trash/hdp-admin/<CheckPointTimeStamp>/zone_name/file1

Related Information

HDFS-8831

Create an HDFS Admin User

How to create an HDFS admin user.

About this task

To capitalize on the capabilities of HDFS data at rest encryption, you will need two separate types of HDFS administrative accounts:

- HDFS administrative user: an account in the hdfs supergroup that is used to manage encryption keys and encryption zones. Examples in this chapter use an administrative user account named encr.
- HDFS service user: the system-level account traditionally associated with HDFS. By default this is user hdfs in HDP. This account owns the HDFS DataNode and NameNode processes.

  Note:

  This is a system-only account. Physical users should not be given access to this account.

Complete the following steps to create a new HDFS administrative user.

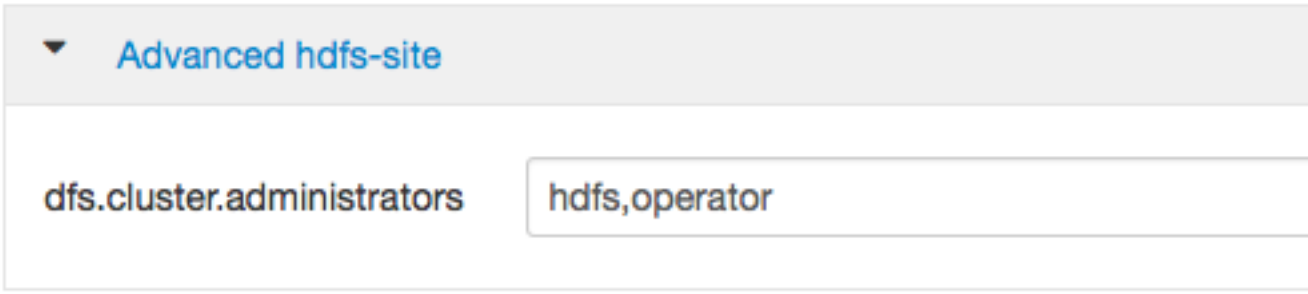
Note: These steps use sample values for group (operator) and user account (opt1).

Procedure

1. Create a new group called operator.
2. Add a new user (for example, opt1) to the group.
3. Add principal opt1@EXAMPLE.COM and create a keytab.
4. Login as opt1, and do a kinit operation.
5. In Ambari, replace the current value of dfs.permissions.superusergroup with the group name “operator”.

   Note:

   You can assign only one administrator group for the dfs.permissions.superusergroup parameter.
6. In Ambari, add hdfs,operator to dfs.cluster.administrators:



7. Add opt1 to the KMS blacklist. Set the corresponding property in Ambari: hadoop.kms.blacklist.DECRYPT_EEK=opt1.
8. Restart HDFS.

What to do next

Validation

Make sure the opt1 account has HDFS administrative access:

hdfs dfsadmin -report

Make sure the opt1 account cannot access encrypted files. For example, if /data/test/file.txt is in an encryption zone, the following command should return an error:

hdfs dfs -cat /data/test/file.txt

Additional Administrative User Accounts

If you plan to use HDFS data at rest encryption with YARN, we recommend that you create a separate administrative user account for YARN administration.

If you plan to use HDFS data at rest encryption with Oozie, refer to the “Oozie” section of this chapter.

Related Information

Configuring Oozie for HDFS Encryption

Configuring HDP Services for HDFS Encryption

Information on configuring HDP services for HDFS Encryption.

HDFS data at rest encryption is supported on the following HDP components:

> **Note:**
>
> You should create a separate Admin user account for HDFS Data at Rest Encryption for each of the following supported components.

- HBase
- Hive
- Hive on Tez
- MapReduce
- Oozie
- Spark
- Sqoop
- Storm
- WebHDFS
- YARN

HDFS data at rest encryption is not supported on the following components:

- Accumulo
- Falcon
- HDP Search

The remainder of this section describes scenarios and access considerations for accessing HDFS-encrypted files from supporting HDP components.

> **Note:**
>
> When HDFS encryption is configured, Spark is able to access data without any further configuration steps or code changes. HDFS encryption is transparent for Spark users.

Configure HBase for HDFS Encryption

How to configure HBase for HDFS encryption.

About this task

HBase stores all of its data under its root directory in HDFS, configured with hbase.rootdir. The only other directory that the HBase service will read or write is hbase.bulkload.staging.dir.

On HDP clusters, hbase.rootdir is typically configured as /apps/hbase/data, and hbase.bulkload.staging.dir is configured as /apps/hbase/staging. HBase data, including the root directory and staging directory, can reside in an encryption zone on HDFS.

The HBase service user needs to be granted access to the encryption key in the Ranger KMS, because it performs tasks that require access to HBase data (unlike Hive or HDFS).

By design, HDFS-encrypted files cannot be bulk-loaded from one encryption zone into another encryption zone, or from an encryption zone into an unencrypted directory. Encrypted files can only be copied. An attempt to load data from one encryption zone into another will result in a copy operation. Within an encryption zone, files can be copied, moved, bulk-loaded, and renamed.

Recommendations

- Make the parent directory for the HBase root directory and bulk load staging directory an encryption zone, instead of just the HBase root directory. This is because HBase bulk load operations need to move files from the staging directory into the root directory.
- In typical deployments, /apps/hbase can be made an encryption zone.
- Do not create encryption zones as subdirectories under /apps/hbase, because HBase may need to rename files across those subdirectories.
- The landing zone for unencrypted data should always be within the destination encryption zone.

Procedure

- On a cluster without HBase currently installed:
  a) Create the /apps/hbase directory, and make it an encryption zone.
  b) Configure hbase.rootdir=/apps/hbase/data.
  c) Configure hbase.bulkload.staging.dir=/apps/hbase/staging.
- On a cluster with HBase already installed, perform the following steps:
  a) Stop the HBase service.
  b) Rename the /apps/hbase directory to /apps/hbase-tmp.
  c) Create an empty /apps/hbase directory, and make it an encryption zone.
  d) DistCp -skipcrccheck -update all data from /apps/hbase-tmp to /apps/hbase, preserving user-group permissions and extended attributes.
  e) Start the HBase service and verify that it is working as expected.
  f) Remove the /apps/hbase-tmp directory.

What to do next

Changes in Behavior after HDFS Encryption is Enabled

The HBase bulk load process is a MapReduce job that typically runs under the user who owns the source data. HBase data files created as a result of the job are then bulk loaded in to HBase RegionServers. During this process, HBase RegionServers move the bulk-loaded files from the user's directory and move (rename) the files into the HBase root directory (/apps/hbase/data). When data at rest encryption is used, HDFS cannot do a rename across encryption zones with different keys.

Workaround: run the MapReduce job as the hbase user, and specify an output directory that resides in the same encryption zone as the HBase root directory.

Configuring Hive for HDFS Encryption

How to configure Hive for HDFS encryption.

Recommendation: Store Hive data in an HDFS path called /apps/hive.

Configure Hive Tables for HDFS Encryption

Before enabling encryption zones, decide whether to store your Hive tables across one zone or multiple encryption zones.

Procedure

For a Single Encryption Zone

1. To configure a single encryption zone for your entire Hive warehouse:
   a) Rename /apps/hive to /apps/hive-old.
   b) Create an encryption zone at /apps/hive.
   c) distcp all of the data from /apps/hive-old to /apps/hive.
2. To configure the Hive scratch directory (hive.exec.scratchdir) so that it resides inside the encryption zone:
   a) Set the directory to /apps/hive/tmp.
   b) Make sure that the permissions for /apps/hive/tmp are set to 1777.

What to do next

For multiple Encryption Zones

To access encrypted databases and tables with different encryption keys, configure multiple encryption zones.

For example, to configure two encrypted tables, ez1.db and ez2.db, in two different encryption zones:

1. Create two new encryption zones, /apps/hive/warehouse/ez1.db and /apps/hive/warehouse/ez2.db.
2. Load data into Hive tables ez1.db and ez2.db as usual, using LOAD statements. (For additional considerations, see "Loading Data into an Encrypted Table.")

Loading Data into an Encrypted Table

By design, HDFS-encrypted files cannot be moved or loaded from one encryption zone into another encryption zone, or from an encryption zone into an unencrypted directory. Encrypted files can only be copied.

Within an encryption zone, files can be copied, moved, loaded, and renamed.

Recommendations:

- When loading unencrypted data into encrypted tables (e.g., LOAD DATA INPATH), we recommend placing the source data (to be encrypted) into a landing zone within the destination encryption zone.
- An attempt to load data from one encryption zone into another will result in a copy operation. Distcp will be used to speed up the process if the size of the files being copied is higher than the value specified by the hive.exec.copyfile.maxsize property. The default limit is 32 MB.

Here are two approaches for loading unencrypted data into an encrypted table:

- To load unencrypted data into an encrypted table, use the LOAD DATA ... statement.

  If the source data does not reside inside the encryption zone, the LOAD statement will result in a copy. If your data is already inside HDFS, though, you can use distcp to speed up the copying process.
- If the data is already inside a Hive table, create a new table with a LOCATION inside an encryption zone, as follows:

  CREATE TABLE encrypted_table [STORED AS] LOCATION ... AS SELECT * FROM <unencrypted_table>

  Note:

  The location specified in the CREATE TABLE statement must be within an encryption zone. If you create a table that points LOCATION to an unencrypted directory, your data will not be encrypted. You must copy your data to an encryption zone, and then point LOCATION to that encryption zone.

If your source data is already encrypted, use the CREATE TABLE statement. Point LOCATION to the encrypted source directory where your data resides:

CREATE TABLE encrypted_table [STORED AS] LOCATION ... AS SELECT * FROM <encrypted_source_directory>

This is the fastest way to create encrypted tables.

Encrypting Other Hive Directories

Reference information on encrypting other Hive directories.

- LOCALSCRATCHDIR : The MapJoin optimization in Hive writes HDFS tables to a local directory and then uploads them to distributed cache. To enable encryption, either disable MapJoin (set hive.auto.convert.join to false) or encrypt the local Hive Scratch directory (hive.exec.local.scratchdir). Performance note: disabling MapJoin will result in slower join performance.
- DOWNLOADED_RESOURCES_DIR : Jars that are added to a user session and stored in HDFS are downloaded to hive.downloaded.resources.dir. If you want these Jar files to be encrypted, configure hive.downloaded.resources.dir to be part of an encryption zone. This directory needs to be accessible to the HiveServer2.
- NodeManager Local Directory List: Hive stores Jars and MapJoin files in the distributed cache, so if you'd like to use MapJoin or encrypt Jars and other resource files, the YARN configuration property NodeManager Local Directory List (yarn.nodemanager.local-dirs) must be configured to a set of encrypted local directories on all nodes.

  Alternatively, to disable MapJoin, set hive.auto.convert.join to false.

Additional Changes in Behavior with HDFS-Encrypted Tables

Additional behavioral changes with HDFS-encrypted tables.

- Users reading data from read-only encrypted tables must have access to a temp directory that is encrypted with at least as strong encryption as the table.
- By default, temp data related to HDFS encryption is written to a staging directory identified by the hive-exec.stagingdir property created in the hive-site.xml file associated with the table folder.
- As of HDP-2.6.0, Hive INSERT OVERWRITE queries require a Ranger URI policy to allow write operations, even if the user has write privilege granted through HDFS policy. To fix the failing Hive INSERT OVERWRITE queries:
  1. Create a new policy under the Hive repository.
  2. In the dropdown where you see Database, select **URI**.
  3. Update the path (Example: /tmp/*).
  4. Add the users and group and save.
  5. Retry the insert query.
- When using encryption with Trash enabled, table deletion operates differently than the default trash mechanism. For more information see “Deleting Files from an Encryption Zone with Trash Enabled”.

Related Information

Deleting Files from an Encryption Zone with Trash Enabled

Configure YARN for HDFS Encryption

How to configure YARN for HDFS encryption.

About this task

Recommendation: Make /apps/history a single encryption zone. History files are moved between the intermediate and done directories, and HDFS encryption will not allow you to move encrypted files across encryption zones.

Procedure

1. On a cluster with MapReduce over YARN installed, create the /apps/history directory and make it an encryption zone.
2. If /apps/history already exists and is not empty:
   a) Create an empty /apps/history-tmp directory.
   b) Make /apps/history-tmp an encryption zone.
   c) Copy (distcp) all data from /apps/history into /apps/history-tmp.
   d) Remove /apps/history.
   e) Rename /apps/history-tmp to /apps/history.

Configuring Oozie for HDFS Encryption

How to configure Oozie for HDFS encryption.

Recommendations

A new Oozie administrator role (oozie-admin) has been created in HDP 2.3.

This role enables role separation between the Oozie daemon and administrative tasks. Both the oozie-admin role and the oozie role must be specified in the adminusers.txt file. This file is installed in HDP 2.3 with both roles specified. Both are also defined in Ambari 2.1 as well. Modification is only required if administrators choose to change the default administrative roles for Oozie.

If oozie-admin is used as the Oozie administrator user in your cluster, then the role is automatically managed by ambari.

If you plan to create an Oozie admin user other than oozie-admin, add the chosen username to adminusers.txt under the $OOZIE_HOME/conf directory.

Here is a sample adminusers.txt file:

```
#
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Users should be set using following rules:
#
#     One user name per line
#     Empty lines and lines starting with '#' are ignored

oozie
oozie-admin
```

Configuring Sqoop for HDFS Encryption

Reference information on configuring HBase for Sqoop encryption.

Following are considerations for using Sqoop to import or export HDFS-encrypted data.

Recommendations

- For Hive:

  Make sure that you are using Sqoop with the --target-dir parameter set to a directory that is inside the Hive encryption zone. Specify the -D option after sqoop import.

  For example:

  ```
  sqoop import \
   -D sqoop.test.import.rootDir=<root-directory> \
   --target-dir <directory-inside-encryption-zone> \
   <additional-arguments>
  ```

- For append or incremental import:

  Make sure that the sqoop.test.import.rootDir property points to the encryption zone specified in the --target-dir argument.

- For HCatalog:

  No special configuration is required.

Configure WebHDFS for HDFS Encryption

How to configure HBase for WebHDFS encryption.

About this task

Recommendations

WebHDFS is supported for writing and reading files to and from encryption zones.

Procedure

To access encrypted files via WebHDFS, complete the following steps:

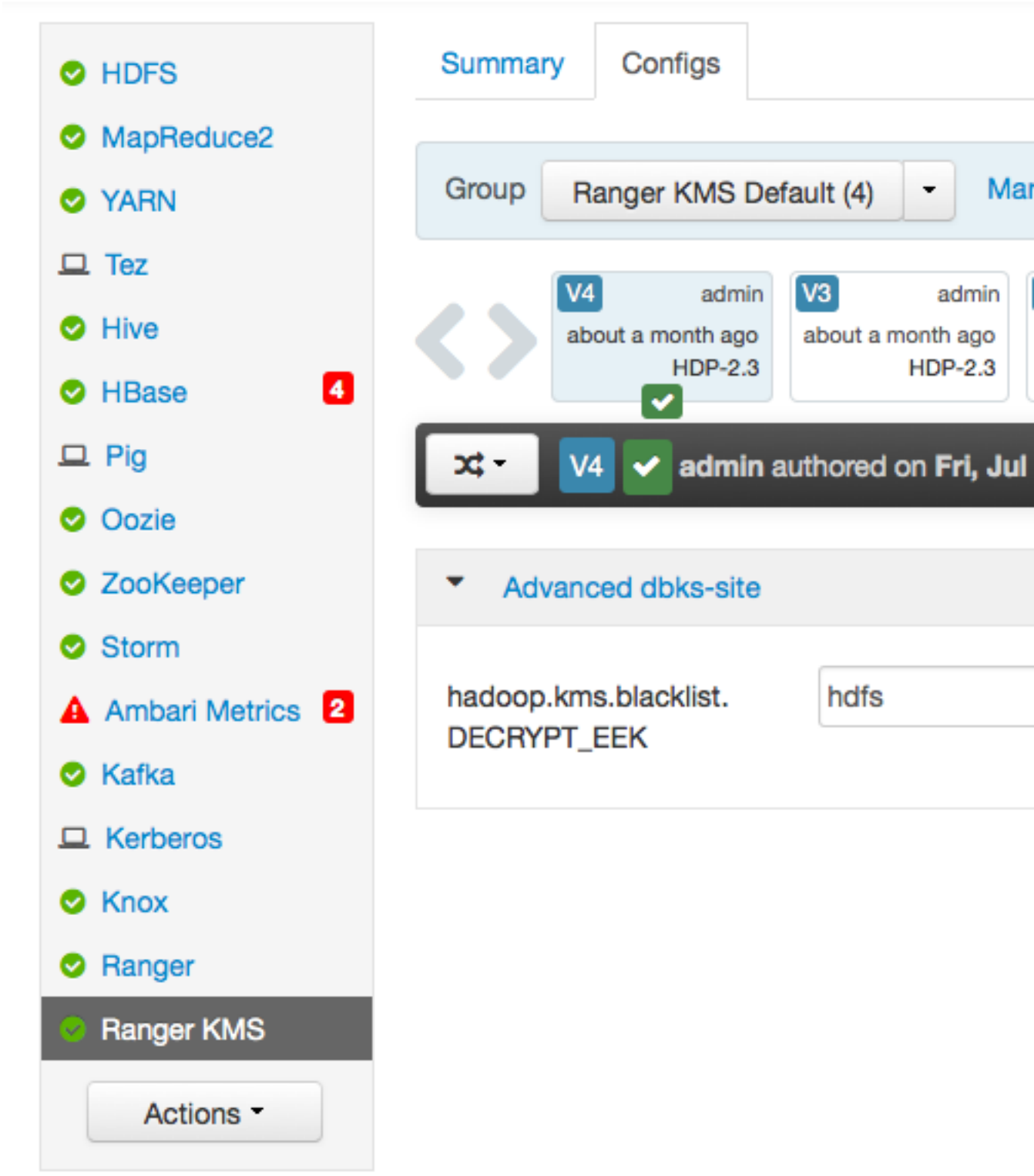
1. To enable WebHDFS in hdfs-site.xml, set the dfs.webhdfs.enabled property to true:

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

2. Make sure that you have separate HDFS administrative and service users, as described in “Create an HDFS Admin User”.
3. Remove the HDFS service user from the blacklist:

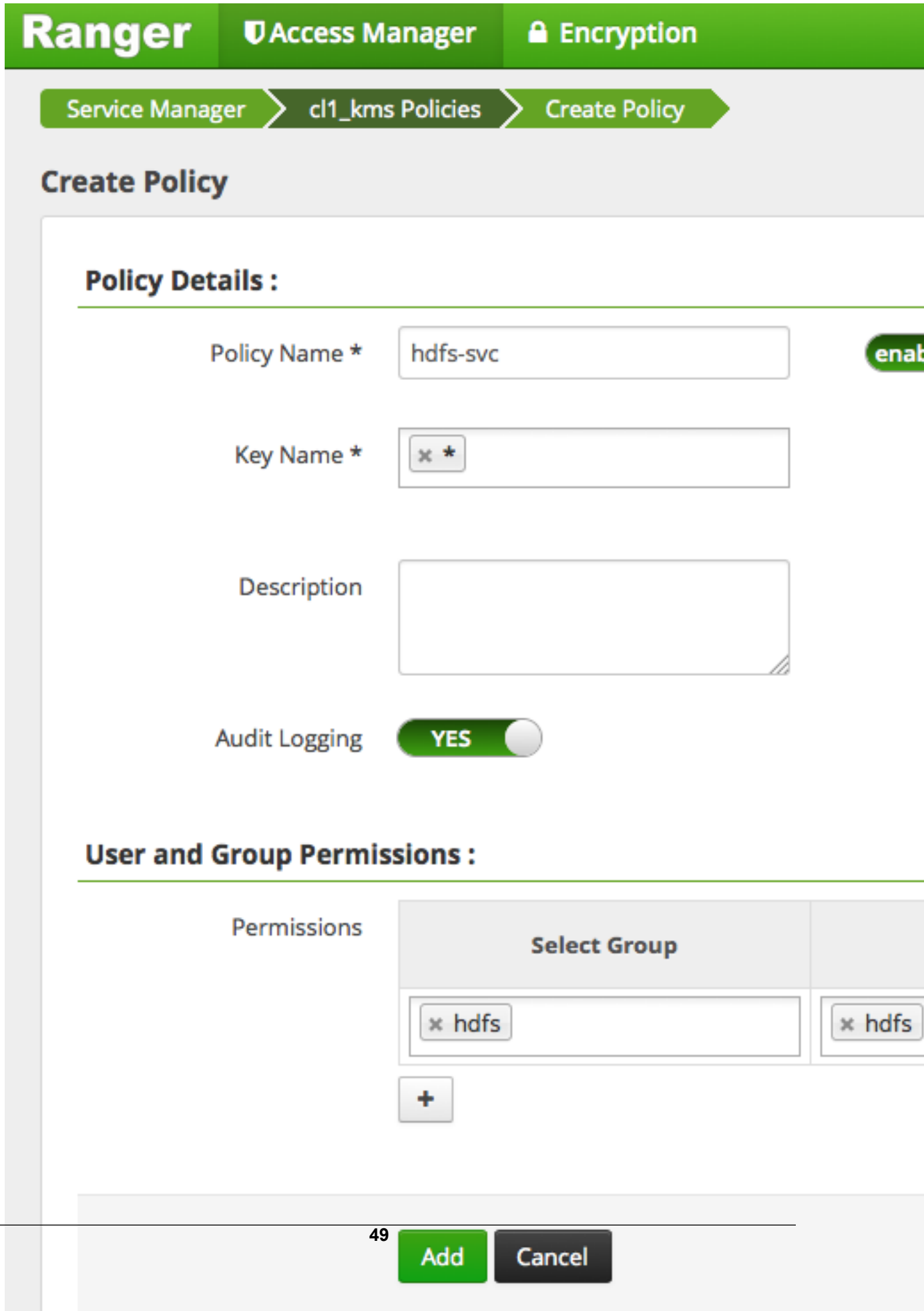
   KMS supports a blacklist and a whitelist for key access (through kms-acls.xml). By default the hdfs service user is included in the blacklist for decrypt_eek operations. To support WebHDFS, the HDFS service user must not be on the key access blacklist.

   a) To edit the blacklist using Ambari, go to Ranger KMS -> Configs, and search for "blacklist" or open the Advanced dbks-site list.
   b) Remove hdfs from the hadoop.kms.blacklist.DECRYPT_EEK property:



c) Restart Ranger KMS.

4. The HDFS service user must have GENERATE_EEK and DECRYPT_EEK permissions. To add the permissions using the Ranger Web UI, select the Access Manager tab-> Resource Based Policies (the default Access Manager view). Select the key store, select the policy, and click the edit icon. In the Permissions column click the edit icon and check the boxes for GenerateEEK and DecryptEEK. Then click Save.


Ranger
Access Manager
Encryption
Service Manager
cl1_kms Policies
Create Policy
Create Policy
Policy Details :
Policy Name *
hdfs-svc
Key Name *
*
Description
Audit Logging
YES
User and Group Permissions :
Permissions
Select Group
hdfs
hdfs
Add
Cancel

5. Because the HDFS service user will have access to all keys, the HDFS service user should not be the administrative user. Specify a different administrative user in hdfs-site.xml for the administrative user.

Related Information

Create an HDFS Admin User

Ranger KMS Administration

Running DataNodes as Non-Root

How to run DataNodes as non-Root.

Historically, part of the security configuration for HDFS involved starting the DataNode as the root user, and binding to privileged ports for the server endpoints. This was done to address a security issue whereby if a MapReduce task was running and the DataNode stopped running, it would be possible for the MapReduce task to bind to the DataNode port and potentially do something malicious. The solution to this scenario was to run the DataNode as the root user and use privileged ports. Only the root user can access privileged ports.

You can now use Simple Authentication and Security Layer (SASL) to securely run DataNodes as a non-root user. SASL is used to provide secure communication at the protocol level.

> **Note:**
>
> Make sure to execute a migration from using root to start DataNodes to using SASL to start DataNodes in a very specific sequence across the entire cluster. Otherwise, there could be a risk of application downtime.

In order to migrate an existing cluster that used root authentication to start using SASL instead, first ensure that HDP 2.2 or later has been deployed to all cluster nodes as well as any external applications that need to connect to the cluster. Only the HDFS client in versions HDP 2.2 and later can connect to a DataNode that uses SASL for authentication of data transfer protocol, so it is vital that all callers have the correct version before migrating. After HDP 2.2 or later has been deployed everywhere, update the configuration of any external applications to enable SASL. If an HDFS client is enabled for SASL, it can connect successfully to a DataNode running with either root authentication or SASL authentication. Changing configuration for all clients guarantees that subsequent configuration changes on DataNodes will not disrupt the applications. Finally, each individual DataNode can be migrated by changing its configuration and restarting. It is acceptable to temporarily have a mix of some DataNodes running with root authentication and some DataNodes running with SASL authentication during this migration period, because an HDFS client enabled for SASL can connect to both.

Configuring DataNode SASL

Use the following steps to configure DataNode SASL to securely run a DataNode as a non-root user

Procedure

1. Shut down the DataNode using the applicable commands in “Controlling HDP Services Manually”.
2. Enable SASL:
   a) Configure the following properties in the /etc/hadoop/conf/hdfs-site.xml file to enable DataNode SASL:

      The dfs.data.transfer.protection property enables DataNode SASL. You can set this property to one of the following values:

      - authentication -- Establishes mutual authentication between the client and the server.
      - integrity -- in addition to authentication, it guarantees that a man-in-the-middle cannot tamper with messages exchanged between the client and the server.

- privacy -- in addition to the features offered by authentication and integrity, it also fully encrypts the messages exchanged between the client and the server.

b) In addition to setting a value for the dfs.data.transfer.protection property, you must set the dfs.http.policy property to HTTPS_ONLY. You must also specify ports for the DataNode RPC and HTTP Servers.

```
<property>
  <name>dfs.data.transfer.protection</name>
  <value>integrity</value>
</property>

<property>
  <name>dfs.datanode.address</name>
  <value>0.0.0.0:10019</value>
</property>

<property>
  <name>dfs.datanode.http.address</name>
  <value>0.0.0.0:10022</value>
</property>

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

3. Update Environment Settings. Edit the following setting in the /etc/hadoop/conf/hadoop-env.sh file, as shown below:

```
#On secure datanodes, user to run the datanode as after dropping
 privileges
export HADOOP_SECURE_DN_USER=
```

The export HADOOP_SECURE_DN_USER=hdfs line enables the legacy security configuration, and must be set to an empty value in order for SASL to be enabled.

4. Start the DataNode services.