Configuring Wire Encryption

Date of Publish: 2018-07-15

http://docs.hortonworks.com

Contents

Wire Encryption

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

:

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

This chapter provides information about configuring and connecting to wire-encrypted components.

Enable RPC Encryption

How to enabke RPC Encryption.

About this task

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

Procedure

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

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

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

Enable Data Transfer Protocol

How to enable Data Transfer Protocol.

About this task

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

Procedure

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

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

rc4 is also supported.

Enabling SSL Understanding the Hadoop SSL Keystore Factory

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

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

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

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

Note:

For more information on JDK SSL Management, see “Using SSL” in “Monitoring and Managing Using JMX Technology” (link below).

Related Information

Java SE Documentation> Monitoring and Management Using JMX Technology> Using SSL

Creating and Managing SSL Certificates

Creating and managing SSL certificates: CA, OpenSSL, installing certificates in the Hadoop SSL keystore factory.

Obtain a Certificate from a Trusted Third Party Certification Authority CA

To obtain a certificate signed by a third-party CA, generate and submit a Certificate Signing Request (CSR) for each cluster node.

About this task

A third-party Certification Authority (CA) accepts certificate requests from entities, authenticates applications, issues certificates, and maintains status information about certificates. Associated cryptography guarantees that a signed certificate is computationally difficult to forge. Thus, as long as the CA is a genuine and trusted authority, clients have high assurance that they are connecting to the machines that they are attempting to connect with.

Procedure

1. From the service user account associated with the component (such as hive, hbase, oozie, or hdfs, shown below as <service_user>), generate the host key: su -l <service_user> -C "keytool -keystore <client-keystore> -genkey -alias <host>".
2. At the prompts, enter the information required by the CSR.

   Note:

   Request generation information and requirements vary depending on the certificate authority. Check with your CA for details.

   Example using default keystore keystore.jks:

   ```
   su -l hdfs -c "keytool -keystore keystore.jks -genkey -alias n3"

   Enter keystore password: ********
   What is your first and last name?
   [Unknown]:  hortonworks.com
   What is the name of your organizational unit?
   [Unknown]:  Development
   What is the name of your organization?
   [Unknown]:  Hortonworks
   What is the name of your City or Locality?
   [Unknown]:  SantaClara
   What is the name of your State or Province?
   [Unknown]:  CA
   What is the two-letter country code for this unit?
   [Unknown]:  US
   Is <CN=hortonworks.com, OU=Development, O=Hortonworks, L=SantaClara,
    ST=CA,
   C=US correct?
   [no]:  yes

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

   By default, keystore uses JKS format for the keystore and truststore. The keystore file is created in the user's home directory. Access to the keystore requires the password and alias.
3. Verify that the key was generated; for example: su -l hdfs -c "keytool -list -v -keystore keystore.jks".

4. Create the CSR file:su -l hdfs -c "keytool -keystore <keystorename> -certreq -alias <host> -keyalg rsa -file <host>.csr".

   This command generates a certificate signing request that can be sent to a CA. The file <host>.csr contains the CSR.

   The CSR is created in the user's home directory.
5. Confirm that the keystore.jks and <host>.csr files exist by running the following command and making sure that the files are listed in the output: su -l hdfs -c "ls ~/".
6. Submit the CSR to your Certificate Authority.
7. To import and install keys and certificates, follow the instructions sent to you by the CA.

Create and Set Up an Internal CA OpenSSL

OpenSSL provides tools to allow you to create your own private certificate authority. How to create and set up a CA.

About this task

Considerations:

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

Note:

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

Before you begin

Install openssl. For example, on CentOS run yum install openssl.

Procedure

1. Generate the key and certificate for a component process.

   The first step in deploying HTTPS for a component process (for example, Kafka broker) is to generate the key and certificate for each node in the cluster. You can use the Java keytool utility to accomplish this task. Start with a temporary keystore, so that you can export and sign it later with the CA.

   Create the key and certificate:

   ```
   $ keytool -keystore <keystore-file> -alias localhost -validity <validity>
    -genkey
   ```

   where:

   <keystore-file> is the keystore file that stores the certificate. The keystore file contains the private key of the certificate; therefore, it needs to be kept safely.

   <validity> is the length of time (in days) that the certificate will be valid.

   Make sure that the common name (CN) matches the fully qualified domain name (FQDN) of the server. The client compares the CN with the DNS domain name to ensure that it is indeed connecting to the desired server, not a malicious server.
2. Create the Certificate Authority (CA)

   After step 1, each machine in the cluster has a public-private key pair and a certificate that identifies the machine. The certificate is unsigned, however, which means that an attacker can create such a certificate to pretend to be any machine.

To prevent forged certificates, it is very important to sign the certificates for each machine in the cluster.

A CA is responsible for signing certificates, and associated cryptography guarantees that a signed certificate is computationally difficult to forge. Thus, as long as the CA is a genuine and trusted authority, the clients have high assurance that they are connecting to the machines that they are attempting to connect with.

Here is a sample openssl command to generate a CA:

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
```

The generated CA is simply a public-private key pair and certificate, intended to sign other certificates.

3. Add the generated CA to the server's truststore: keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert.
4. Add the generated CA to the client's truststore, so that clients know that they can trust this CA: keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert.

   In contrast to the keystore in step 1 that stores each machine's own identity, the truststore of a client stores all of the certificates that the client should trust. Importing a certificate into one's truststore also means trusting all certificates that are signed by that certificate.

   Trusting the CA means trusting all certificates that it has issued. This attribute is called a "chain of trust," and is particularly useful when deploying SSL on a large cluster. You can sign all certificates in the cluster with a single CA, and have all machines share the same truststore that trusts the CA. That way all machines can authenticate all other machines.
5. Sign all certificates generated in Step 1 with the CA generated in Step 2:
   a) Export the certificate from the keystore: keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file.
   b) Sign the certificate with the CA: openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days <validity> -CAcreateserial -passin pass:<ca-password>.
6. Import the CA certificate and the signed certificate into the keystore. For example:

```
$ keytool -keystore server.keystore.jks -alias CARoot -import -file ca-
cert
$ keytool -keystore server.keystore.jks -alias localhost -import -file
 cert-signed
```

The parameters are defined as follows:

Parameter	Description
keystore	The location of the keystore
ca-cert	The certificate of the CA
ca-key	The private key of the CA
ca-password	The passphrase of the CA
cert-file	The exported, unsigned certificate of the server
cert-signed	The signed certificate of the server

7. All of the preceding steps can be placed into a bash script.

   In the following example, note that one of the commands assumes a password of test1234. Specify your own password before running the script.

```
#!/bin/bash

#Step 1
keytool -keystore server.keystore.jks -alias localhost -validity 365 -
genkey
```

```
#Step 2
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
keytool -keystore server.truststore.jks -alias CARoot -import -file ca-
cert
keytool -keystore client.truststore.jks -alias CARoot -import -file ca-
cert

#Step 3
keytool -keystore server.keystore.jks -alias localhost -certreq -file
 cert-file
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed
 -days 365 -CAcreateserial -passin pass:test1234
keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore server.keystore.jks -alias localhost -import -file cert-
signed
```

8. Set up the CA directory structure:mkdir -m 0700 /root/CA /root/CA/certs /root/CA/crl /root/CA/newcerts /root/CA/private.
9. Move the CA key to /root/CA/private and the CA certificate to /root/CA/certs.mv ca.key /root/CA/private;mv ca.crt /root/CA/certs.
10. Add required files:touch /root/CA/index.txt; echo 1000 >> /root/CA/serial.
11. Set permissions on the ca.key:chmod 0400 /root/ca/private/ca.key.
12. Open the OpenSSL configuration file:vi /etc/pki/tls/openssl.cnf.
13. Change the directory paths to match your environment:

```
[ CA_default ]

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

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

x509_extensions = usr_cert              # The extensions to add to the
 cert
```

14. Save the changes and restart OpenSSL.

Example

Example of setting up an OpenSSL internal CA:

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

Generating RSA private key, 8192 bit long modulus
```

```
.....................................................................................
++
.....................++
e is 65537 (0x10001)
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:California
Locality Name (eg, city) [Default City]:SantaClara
Organization Name (eg, company) [Default Company Ltd]:Hortonworks
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:nn
Email Address []:it@hortonworks.com

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

Install Certificates in the Hadoop SSL Keystore Factory (HDFS, MapReduce, and YARN)

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

Procedure

1. Create a directory for the server and client stores.mkdir -p <SERVER_KEY_LOCATION> ; mkdir -p <CLIENT_KEY_LOCATION>.
2. Import the server certificate from each node into the HTTP Factory truststore.cd <SERVER_KEY_LOCATION> ; keytool -import -noprompt -alias <remote-hostname> -file <remote-hostname>.jks -keystore <TRUSTSTORE_FILE> -storepass <SERVER_TRUSTSTORE_PASSWORD>.
3. Create a single truststore file containing the public key from all certificates, by importing the public key for each CA or from each self-signed certificate pair: keytool -import -noprompt -alias <host> -file $CERTIFICATE_NAME -keystore <ALL_JKS> -storepass <CLIENT_TRUSTSTORE_PASSWORD>.
4. Copy the keystore and truststores to every node in the cluster.
5. Validate the common truststore file on all hosts. keytool -list -v -keystore <ALL_JKS> -storepass <CLIENT_TRUSTSTORE_PASSWORD>.
6. Set permissions and ownership on the keys:

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

Note:

The complete path of the <SERVER_KEY_LOCATION> and the <CLIENT_KEY_LOCATION> from the root directory /etc must be owned by the yarn user and the hadoop group.

Installing Certificates in the Hadoop SSL Keystore Factory Using a CA Signed Certificate

How to use a CA-signed certificate.

Procedure

1. Run the following command to create a self-signing rootCA and import the rootCA into the client truststore. This is a private key; it should be kept private. The following command creates a 2048-bit key: openssl genrsa -out <clusterCA>.key 2048.
2. Self-sign the rootCA. The following command signs for 300 days. It will start an interactive script that requests name and location information.openssl req -x509 -new -key <clusterCA>.key -days 300 -out <clusterCA>.
3. Import the rootCA into the client truststore:keytool -importcert -alias <clusterCA> -file $clusterCA -keystore <clustertruststore> -storepass <clustertruststorekey>.

   Note:

   Make sure that the ssl-client.xml file on every host is configured to use this $clustertrust store.

   When configuring with Hive point to this file; when configuring other services install the certificate in the Java truststore.
4. For each host, sign the certreq file with the rootCA:openssl x509 -req -CA $clusterCA.pem -CAkey <clusterCA>.key -in <host>.cert -out $host.signed -days 300 -CAcreateserial.
5. On each host, import the rootCA and the signed cert back in:

```
keytool -keystore <hostkeystore> -storepass <hoststorekey> -alias
 <clusterCA> -import -file cluster1CA.pem
keytool -keystore <hostkeystore> -storepass <hoststorekey> -alias
 `hostname -s` -import -file <host>.signed -keypass <hostkey>
```

Enabling SSL for HDP Components

How to enable SSL on specific HDP components.

The following table contains links to instructions for enabling SSL on specific HDP components.

- Hadoop
- MapReduce
- YARN
- Oozie
- HBase
- Hive (HiveServer2)
- Kafka
- Ambari Server
- Falcon
- Sqoop
- Knox Gateway
- Flume

- Accumulo
- Phoenix

Enable SSL for WebHDFS, MapReduce, Tez, and YARN

This section explains how to set up SSL for WebHDFS, YARN and MapReduce. Before you begin, make sure that the SSL certificate is properly configured, including the keystore and truststore that will be used by WebHDFS, MapReduce, and YARN.

About this task

HDP supports the following SSL modes:

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

Procedure

1. Set the following property values (or add the properties if required) in core-site.xml:

```
hadoop.ssl.require.client.cert=false
hadoop.ssl.hostname.verifier=DEFAULT
hadoop.ssl.keystores.factory.class=org.apache.hadoop.security.ssl.FileBasedKeyStores
hadoop.ssl.server.conf=ssl-server.xml
hadoop.ssl.client.conf=ssl-client.xml
```

Note:

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

2. Set the following properties (or add the properties if required) in hdfs-site.xml:

```
dfs.http.policy=<Policy>
dfs.client.https.need-auth=true (optional for mutual client/server
 certificate validation)
dfs.datanode.https.address=<hostname>:50475
dfs.namenode.https-address=<hostname>:50470
```

Where <Policy> is either:

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

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

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

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

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

5. Create an ssl-server.xml file for the Hadoop SSL Keystore Factory:
   a) Copy the example SSL Server configuration file and modify the settings for your environment:cp /etc/hadoop/conf/ssl-server.xml.example /etc/hadoop/conf/ssl-server.xml.
   b) Configure the server SSL properties:

      Configuration Properties in ssl-server.xml

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

      For example:

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

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

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

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

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

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

<property>
```

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

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

6. Create an ssl-client.xml file for the Hadoop SSL Keystore Factory:
   a) Copy the client truststore example file:

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

   b) Configure the client trust store values:

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

7. Set the following properties in the tez-site.xml file:

```
tez.runtime.shuffle.ssl.enable=true
tez.runtime.shuffle.keep-alive.enabled=true
```

8. Copy the configuration files (core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml, ssl-server.xml, tez-site.xml and ssl-client.xml), including the ssl-server and ssl-client store files if the Hadoop SSL Keystore Factory uses it's own keystore and truststore files, to all nodes in the cluster.
9. Restart services on all nodes in the cluster.

Enable SSL for HttpFS

How to configure HttpFS to work over SSL.

Procedure

1. Edit the httpfs-env.sh script in the configuration directory:

```
HTTPFS_SSL_ENABLED=true
HTTPFS_SSL_KEYSTORE_FILE=$HOME/.keystore
HTTPFS_SSL_KEYSTORE_PASS=password
```

2. In the HttpFS tomcat/conf directory, replace the server.xml file with the ssl-server.xml file.
3. Create an SSL certificate for the HttpFS server. As the httpfs Unix user, use the Java keytool command to create the SSL certificate:$ keytool -genkey -alias tomcat -keyalg RSA.

   You will be asked a series of questions in an interactive prompt. It will create the keystore file, which will be named .keystore and located in the httpfs user home directory.

   The password you enter for “keystore password” must match the value of the HTTPFS_SSL_KEYSTORE_PASS environment variable set in the httpfs-env.sh script in the configuration directory.

   The answer to “What is your first and last name?” (i.e. “CN”) must be the host name of the machine where the HttpFS Server will be running.
4. Start HttpFS. It should work over HTTPS.

5. Utilizing the Hadoop FileSystem API or the Hadoop FS shell, use the swebhdfs:// scheme. Make sure the JVM is picking up the truststore containing the public key of the SSL certificate if you are using a self-signed certificate.

Related Information

HttpFS

Enable SSL on Oozie

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

Procedure

1. If Oozie server is running, stop Oozie.
2. Change the Oozie environment variables for HTTPS if required:
   - OOZIE_HTTPS_PORT set to Oozie HTTPS port. The default value is 11443.
   - OOZIE_HTTPS_KEYSTORE_FILE set to the keystore file that contains the certificate information. Default value $<HOME>/.keystore, that is the home directory of the Oozie user.
   - OOZIE_HTTPS_KEYSTORE_PASS set to the password of the keystore file. Default value password.

   Note:

   See “Oozie Environment Setup” (link below) for more details.
3. Run the following command to enable SSL on Oozie:su -l oozie -c "/usr/hdp/current/oozie-server/bin/oozie-setup.sh prepare-war -secure".
4. Start the Oozie server.

Related Information

Apache Oozie Documentation> Oozie Environment Setup

Configure the Oozie Client to Connect Using SSL

Use the following procedure to configure the Oozie client to connect using SSL. The first two steps are only necessary if you are using a self-signed Certificate. Also, these steps must be performed on every machine on which you intend to use the Oozie Client.

Procedure

1. Copy or download the .cert file onto the client machine.
2. Run the following command (as root) to import the certificate into the JRE keystore. This will allow any Java program, including the Oozie client, to connect to the Oozie Server using the self-signed certificate.sudo keytool -import -alias tomcat -file path/to/certificate.cert -keystore ${JRE_cacerts}.

   Where ${JRE_cacerts} is the path to the JRE .certs file. Its location may differ depending on the operating system, but its typically named cacerts and is located at ${JAVA_HOME}/lib/security/cacerts, but it may be in a different directory under ${JAVA_HOME} (you may want to create a backup copy of this file first). The default password is changeit.
3. When using the Oozie Client, you must use https://oozie.server.hostname:11443/oozie rather than http://oozie.server.hostname:11000/oozie -- Java will not automatically redirect from the HTTP address to the HTTPS address.

Connect to the Oozie Web UI Using SSL

Use https://oozie.server.hostname:11443/oozie to connect to the Oozie web UI using SSL, but most browsers should redirect if you use http://oozie.server.hostname:11000/oozie.

Note:

If you are using a self-signed certificate, the browser will warn you that it cannot verify the certificate. You will probably need to add the certificate as an exception.

Configure Oozie HCatalogJob Properties

How to configure Oozie HCatalogJob properties.

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

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

Enable SSL on the HBase REST Server

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

Procedure

1. Create and install an SSL certificate for HBase, for example to use a self-signed certificate:
   a) Create an HBase keystore: su -l hbase -c "keytool -genkey -alias hbase -keyalg RSA -keysize 1024 -keystore hbase.jks".
   b) At the keytool command prompt:
      - Enter the key password
      - Enter the keystore password

      Note:

      Add these two specified values to the corresponding properties in hbase-site.xml in step 2.
   c) Export the certificate: su -l hbase -c "keytool -exportcert -alias hbase -file certificate.cert -keystore hbase.jks".
   d) (Optional) Add certificate to the Java keystore:

   - If you are not root run: sudo keytool -import -alias hbase -file certificate.cert -keystore /usr/jdk64/jdk1.7.0_45/jre/lib/security/cacerts
   - If you are root: keytool -import -alias hbase -file certificate.cert -keystore /usr/jdk64/jdk1.7.0_45/jre/lib/security/cacerts
2. Add the following properties to the hbase-site.xml configuration file on each node in your HBase cluster:

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

<property>
    <name>hbase.rest.ssl.keystore.store</name>
    <value>/path/to/keystore</value>
</property>

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

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

3. Restart all HBase nodes in the cluster.

Enable SSL on the HBase Web UI

How to enable SSL and TLS on an HBase Web UI.

Procedure

1. Create and install an SSL certificate for HBase, for example to use a self-signed certificate:
   a) Create an HBase keystore: su -l hbase -c "keytool -genkey -alias hbase -keyalg RSA -keysize 1024 -keystore hbase.jks".
   b) At the keytool command prompt:
      - Enter the key password
      - Enter the keystore password

      Note:

      Add these two specified values to the corresponding properties in hbase-site.xml in step 2.
   c) Export the certificate: su -l hbase -c "keytool -exportcert -alias hbase -file certificate.cert -keystore hbase.jks".
   d) (Optional) Add certificate to the Java keystore:

   - If you are not root run: sudo keytool -import -alias hbase -file certificate.cert -keystore /usr/jdk64/jdk1.7.0_45/jre/lib/security/cacerts
   - If you are root: keytool -import -alias hbase -file certificate.cert -keystore /usr/jdk64/jdk1.7.0_45/jre/lib/security/cacerts
2. Add the following properties to the hbase-site.xml configuration file on each node in your HBase cluster:

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

<property>
   <name>hadoop.ssl.enabled</name>
   <value>true</value>
</property>

<property>
    <name>ssl.server.keystore.keypassword</name>
    <value>key-password</value>
</property>

<property>
    <name><ssl.server.keystore.password</name>
    <value>keystore-password</value>
</property>

<property>
    <name>ssl.server.keystore.location</name>
    <value>/tmp/server-keystore.jks</value>
</property>
```

3. Restart all HBase nodes in the cluster.

Enable SSL on HiveServer2

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

About this task

Perform the following steps on the HiveServer2.

Procedure

1. Log into the cluster as the hive user. Having hive user permissions when creating the Java keystore file sets up the proper user::group ownership, which allows HiveServer to access the file and prevents HiveServer startup failure.
2. Run the following command to create a keystore for hiveserver2: keytool -genkey -alias hbase -keyalg RSA -keysize 1024 -keystore hive.jks.
3. Edit the hive-site.xml, set the following properties to enable SSL:

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

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

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

Note:

When hive.server2.transport.mode is binary and hive.server2.authentication is KERBEROS, SSL encryption does not currently work. Set hive.server2.thrift.sasl.qop to auth-conf to enable encryption

4. On the client-side, specify SSL settings for Beeline or JDBC client as follows:jdbc:hive2://<host>:<port>/<database>;ssl=true;sslTrustStore=<path-to-truststore>;trustStorePassword=<password>.

Set up SSL with self-signed certificates

Use the following steps to create and verify self-signed SSL certificates for use with HiveServer2.

Procedure

1. List the keystore entries to verify that the certificate was added. Note that a keystore can contain multiple such certificates: keytool -list -keystore keystore.jks.
2. Export this certificate from keystore.jks to a certificate file: keytool -export -alias example.com -file example.com.crt -keystore keystore.jks.
3. Add this certificate to the client's truststore to establish trust: keytool -import -trustcacerts -alias example.com -file example.com.crt -keystore truststore.jks.
4. Verify that the certificate exists in truststore.jks: keytool -list -keystore truststore.jks.
5. Then start HiveServer2, and try to connect with beeline using: jdbc:hive2://<host>:<port>/<database>;ssl=true;sslTrustStore=<path-to-truststore>;trustStorePassword=<truststore-password>.

Selectively disable SSL protocol versions

To disable specific SSL protocol versions, use the following steps.

Procedure

1. Run openssl ciphers -v (or the corresponding command if not using openssl) to view all protocol versions.

2. In addition to 1, an additional step of going over the HiveServer2 logs may be required to see all the protocols that the node running HiveServer2 is supporting. For that, search for "SSL Server Socket Enabled Protocols:" in the HiveServer2 log file.
3. Add all the SSL protocols that need to be disabled to hive.ssl.protocol.blacklist. Ensure that the property in hiveserver2-site.xml does not override that in hive-site.xml.

Enable SSL for Kafka Clients

Kafka allows clients to connect over SSL. By default SSL is disabled, but it can be enabled as needed.

Before you begin, be sure to generate the key, SSL certificate, keystore, and truststore that will be used by Kafka.

Configuring the Kafka Broker

The Kafka Broker supports listening on multiple ports and IP addresses. To enable this feature, specify one or more comma-separated values in the listeners property in server.properties.

Procedure

1. Both PLAINTEXT and SSL ports are required if SSL is not enabled for inter-broker communication (see the following subsection for information about enabling inter-broker communication): listeners=PLAINTEXT://host.name:port,SSL://host.name:port.

   The following SSL configuration settings are needed on the broker side:

   ```
   ssl.keystore.location = /var/private/ssl/kafka.server.keystore.jks
   ssl.keystore.password = test1234
   ssl.key.password = test1234
   ssl.truststore.location = /var/private/ssl/kafka.server.truststore.jks
   ssl.truststore.password = test1234
   ```

   The following optional settings are available:

Property	Description	Value(s)
ssl.client.auth	Specify whether client authentication is required, requested, or not required. none: no client authentication. required: client authentication is required. requested: client authentication is requested, but a client without certs can still connect. Note: If you set ssl.client.auth to requested or required, then you must provide a truststore for the Kafka broker. The truststore should contain all CA certificates that are used to sign clients' keys.	none
ssl.cipher.suites	Specify one or more cipher suites: named combinations of authentication, encryption, MAC and key exchange algorithms used to negotiate the security settings for a network connection using the TLS or SSL network protocol.	
ssl.enabled.protocols	Specify the SSL protocols that you will accept from clients. Note: SSL is deprecated; its use in production is not recommended.	TLSv1.2,TLSv1.1,TLSv1
ssl.keystore.type	Specify the SSL keystore type.	JKS
ssl.truststore.type	Specify the SSL truststore type.	JKS

2. To enable SSL for inter-broker communication, add the following setting to the broker properties file (default is PLAINTEXT): security.inter.broker.protocol = SSL.
3. To enable any cipher suites other than the defaults that come with JVM (see “Java Cryptography documentation”), you will need to install JCE Unlimited Strength Policy files (download link below).
4. Validate the configuration. After you start the broker, you should see the following information in the server.log file:

```
with addresses: PLAINTEXT -> EndPoint(192.168.64.1,9092,PLAINTEXT),SSL ->
 EndPoint(192.168.64.1,9093,SSL)
```

5. To make sure that the server keystore and truststore are set up properly, run the following command:openssl s_client -debug -connect localhost:9093 -tls1.

   (Note: TLSv1, TLSv1.1, and TLSv1.2 should be listed under ssl.enabled.protocols)

   In the openssl output you should see the server certificate; for example:

```
Server certificate
-----BEGIN CERTIFICATE-----
MIID+DCCAuACCQCx2Rz1tXx3NTANBgkqhkiG9w0BAQsFADB6MQswCQYDVQQGEwJV
UzELMAkGA1UECAwCQ0ExFDASBgNVBAcMC1NhbnRhIENsYXJhMQwwCgYDVQQKDANv
cmcxDDAKBgNVBAsMA29yZzEOMAwGA1UEAwwFa2FmYWsxHDAaBgkqhkiG9w0BCQEW
DXRlc3RAdGVzdC5jb20wHhcNMTUwNzMwMDQyOTMwWhcNMTYwNzI5MDQyOTMwWjBt
MQswCQYDVQQGEwJVUzELMAkGA1UECBMCQ0ExFDASBgNVBAcTC1NhbnRhIENsYXJh
MQwwCgYDVQQKEwNvcmcxDDAKBgNVBAsTA29yZzEfMB0GA1UEAxMWU3JpaGFyc2hh
IENoaW50YWxhcGFuaTCCAbcwggEsBgcqhkjOOAQBMIIBHwKBgQD9f1OBHXUSKVLf
Spwu7OTn9hG3UjzvRADDHj+AtlEmaUVdQCJR+1k9jVj6v8X1ujD2y5tVbNeBO4Ad
NG/yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophUPBPuD9tPFHsMCNVQT
WhaRMvZ1864rYdcq7/IiAxmd0UgBxwIVAJdgUI8VIwvMspK5gqLrhAvwWBz1AoGB
APfhoIXWmz3ey7yrXDa4V7l5lK+7+jrqgvlXTAs9B4JnUVlXjrrUWU/mcQcQgYC0
SRZxI+hMKBYTt88JMozIpuE8FnqLVHyNKOCjrh4rs6Z1kW6jfwv6ITVi8ftiegEk
O8yk8b6oUZCJqIPf4VrlnwaSi2ZegHtVJWQBTDv+z0kqA4GEAAKBgB+Pdz0306bq
TpUAdb2FERMPLFsx06H0x+TULivcp7HbS5yrkV9bXZmv/FD98x76QxXrOq1WpQhY
YDeGDjH+XQkJ6ZxBVBZNJDIpCnfQpfzXAvryQ+cm8oXUsKidtHf4pLMYViXX6BWX
Oc2hX4rG+lC8/NXW+1zVvCr9To9fngzjMA0GCSqGSIb3DQEBCwUAA4IBAQBfyVse
RJ+ugiNlWg5trZscqH0tlocbnek4UuV/xis2eAu9l4EFOM5kRt5GmkGZRcM/zHF8
BRJwXbf0fytMQKSPFk8R4/NGDOlzoK+F7uXeJ0S2u/T29xk0u2i4tjvleq6OCphE
i9vdjM0E0Whf9SHRhOXirOYFX3cL775XwKdzKKRkk+AszFR+mRu90rdoaepQtgGh
9Kfwr4+6AU/dPtdGuomtBQqMxCzlrLd8EYhVVQ97wHIZ3sPvlM5PIhOJ/YHSBJIC
75eo/4acDxZ+j3sR5kcFulzYwFLgDYBaKH/w3mYCgTALeB1zUkX53NVizIvhUd69
XJO4lDSDtGOlfort
-----END CERTIFICATE-----
subject=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=JBrown
issuer=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=kafak/
emailAddress=test@test.com
```

What to do next

If the certificate does not display, or if there are any other error messages, then your keystore is not set up properly.

Related Information

Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7 Download

Java Cryptography Architecture Oracle Providers Documentation for Java Platform Standard Edition 7

Configuring Kafka Producer and Kafka Consumer

Examples for configuring Kafka Producer and Kafka consumer. SSL is supported for new Kafka Producers and Consumer processes; the older API is not supported. Configuration settings for SSL are the same for producers and consumers.

If client authentication is not needed in the broker, then the following is a minimal configuration example:

```
security.protocol = SSL
ssl.truststore.location = /var/private/ssl/kafka.client.truststore.jks
ssl.truststore.password = test1234
```

If client authentication is required, first create a keystore (described earlier in this chapter). Next, specify the following settings:

```
ssl.keystore.location = /var/private/ssl/kafka.client.keystore.jks
ssl.keystore.password = test1234
ssl.key.password = test1234
```

One or more of the following optional settings might also be needed, depending on your requirements and the broker configuration:

Property	Description	Value(s)
ssl.provider	The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.	
ssl.cipher.suites	Specify one or more cipher suites: named combinations of authentication, encryption, MAC and key exchange algorithms used to negotiate the security settings for a network connection using the TLS or SSL network protocol.	
ssl.enabled.protocols	List at least one of the protocols configured on the broker side. Note: SSL is deprecated; its use in production is not recommended.	TLSv1.2,TLSv1.1,TLSv1
ssl.keystore.type	Specify the SSL keystore type.	JKS
ssl.truststore.type	Specify the SSL truststore type.	JKS

The following two examples launch console-producer and console-consumer processes:

```
kafka-console-producer.sh --broker-list localhost:9093 --topic test --
producer.config client-ssl.properties

kafka-console-consumer.sh --bootstrap-server localhost:9093 --topic test --
new-consumer --consumer.config client-ssl.properties
```

Enable SSL for Accumulo

One of the major features added in Accumulo 1.6.0 was the ability to configure Accumulo so that the Thrift communications will run over SSL. Apache Thrift is the remote procedure call library that is leveraged for both intra-server and client communication with Accumulo. Issuing these calls over a secure socket ensures that unwanted actors cannot inspect the traffic sent across the wire. Given the sometimes sensitive nature of data stored in Accumulo and the authentication details for users, secure communications are critical.

Due to the complex and deployment-specific nature of the security model for some systems, Accumulo expects users to provide their own certificates, guaranteeing that they are, in fact, secure. However, for those who require security but do not already operate within the confines of an established security infrastructure, OpenSSL and the Java keytool command can be used to generate the necessary components to enable wire encryption.

To enable SSL with Accumulo, it is necessary to generate a certificate authority and certificates that are signed by that authority. Typically, each client and server has its own certificate, which provides the finest level of control over a secure cluster when the certificates are properly secured.

Related Information

Apache Thrift

Generating a Certificate Authority

How to generate a certificate authority (CA) when enabling SSL for Accumulo.

The certificate authority (CA) controls what certificates can be used to authenticate with each other. To create a secure connection with two certificates, each certificate must be signed by a certificate authority in the "truststore" (A Java KeyStore which contains at least one Certificate Authority's public key). When creating your own certificate authority, a single CA is typically sufficient (and would result in a single public key in the truststore). Alternatively, a third party can also act as a certificate authority (to add an additional layer of security); however, these are typically not a free service.

The following is an example of creating a certificate authority and adding its public key to a Java KeyStore to provide to Accumulo.

```
# Create a private key
openssl genrsa -des3 -out root.key 4096

# Create a certificate request using the private key
openssl req -x509 -new -key root.key -days 365 -out root.pem

# Generate a Base64-encoded version of the PEM just created
openssl x509 -outform der -in root.pem -out root.der

# Import the key into a Java KeyStore
keytool -import -alias root-key -keystore truststore.jks -file root.der

# Remove the DER formatted key file (as we don't need it anymore)
rm root.der
```

Remember to protect root.key and never distribute it, as the private key is the basis for your circle of trust.
The keytool command will prompt you about whether or not the certificate should be trusted: enter "yes". The truststore.jks file, a "truststore", is meant to be shared with all parties communicating with one another. The password provided to the truststore verifies that the contents of the truststore have not been tampered with.

Generating a Certificate Keystore Per Host

How to generate a certificate keystore per host when enabling SSL for Accumulo.

It is desirable to generate a certificate for each host in the system. Additionally, each client connecting to the Accumulo instance running with SSL should be issued its own certificate. Issuing individual certificates to each entity provides proper control to revoke/reissue certificates to clients as necessary, without widespread interruption.

The following commands create a private key for the server, generate a certificate signing request created from that private key, use the certificate authority to generate the certificate using the signing request. and then create a Java KeyStore with the certificate and the private key for our server.

```
# Create the private key for our server
openssl genrsa -out server.key 4096

# Generate a certificate signing request (CSR) with our private key
openssl req -new -key server.key -out server.csr

# Use the CSR and the CA to create a certificate for the server (a reply to
 the CSR)
openssl x509 -req -in server.csr -CA root.pem -CAkey root.key -
CAcreateserial -out server.crt -days 365

# Use the certificate and the private key for our server to create PKCS12
 file
```

```
openssl pkcs12 -export -in server.crt -inkey server.key -certfile server.crt
 -name 'server-key' -out server.p12

# Create a Java KeyStore for the server using the PKCS12 file (private key)
keytool -importkeystore -srckeystore server.p12 -srcstoretype pkcs12 -
destkeystore server.jks -deststoretype JKS

# Remove the PKCS12 file as we don't need it
rm server.p12

# Import the CA-signed certificate to the keystore
keytool -import -trustcacerts -alias server-crt -file server.crt -keystore
 server.jks
```

This, combined with the truststore, provides what is needed to configure Accumulo servers to run over SSL. The private key (server.key), the certificate signed by the CA (server.pem), and the keystore (server.jks) should be restricted to only be accessed by the user running Accumulo on the host it was generated for. Use chown and chmod to protect the files, and do not distribute them over non-secure networks.

Configure Accumulo Servers

Now that the Java KeyStores have been created with the necessary information, the Accumulo configuration must be updated so that Accumulo creates the Thrift server over SSL instead of a normal socket.

Configure the following properties in accumulo-site.xml:

```
<property>
  <name>rpc.javax.net.ssl.keyStore</name>
  <value>/path/to/server.jks</value>
</property>
<property>
  <name>rpc.javax.net.ssl.keyStorePassword</name>
  <value>server_password</value>
</property>
<property>
  <name>rpc.javax.net.ssl.trustStore</name>
  <value>/path/to/truststore.jks</value>
</property>
<property>
  <name>rpc.javax.net.ssl.trustStorePassword</name>
  <value>truststore_password</value>
</property>
<property>
  <name>instance.rpc.ssl.enabled</name>
  <value>true</value>
</property>
```

The keystore and truststore paths are both absolute paths on the local file system (not HDFS). Remember that the server keystore should only be readable by the user running Accumulo and, if you place plain-text passwords in accumulo-site.xml, make sure that accumulo-site.xml is also not globally readable. To keep these passwords out of accumulo-site.xml, consider configuring your system with the new Hadoop CredentialProvider class.

Also, be aware that if unique passwords are used for each server when generating the certificate, this will result in different accumulo-site.xml files for each host. Unique configuration files for each host will add complexity to the configuration management of your instance. The use of a CredentialProvider (a feature from Hadoop which allows for acquisitions of passwords from alternate systems) can help alleviate the issues with unique accumulo-site.xml files on each host. A Java KeyStore can be created using the CredentialProvider tools, which eliminates the need for passwords to be stored in accumulo-site.xml, and can instead point to the CredentialProvider URI which is consistent across hosts.

Configure Accumulo Clients

How to configure Accumulo clients when enabling SSL for Accumulo.

To configure Accumulo clients, use $HOME/.accumulo/config. This is a simple Java properties file: each line is a configuration, key, and value separated by a space, and lines beginning with a # symbol are ignored. For example, if we generated a certificate and placed it in a keystore (as described above), we would generate the following file for the Accumulo client.

```
instance.rpc.ssl.enabled true
rpc.javax.net.ssl.keyStore  /path/to/client-keystore.jks
rpc.javax.net.ssl.keyStorePassword  client-password
rpc.javax.net.ssl.trustStore  /path/to/truststore.jks
rpc.javax.net.ssl.trustStorePassword  truststore-password
```

When creating a ZooKeeperInstance, the implementation will automatically look for this configuration file and set up a connection with the methods defined in this file. The ClientConfiguration class also contains methods that can be used instead of a configuration file on the file system. Again, the paths to the keystore and truststore are on the local file system, not HDFS.

Related Information

Java properties file

Enable SSL for Apache Atlas

This section describes how to enable SSL for Apache Atlas on an Ambari cluster.

Configuring Apache Atlas SSL

Use the following steps to enable Apache Atlas SSL. Both one-way (server authentication) and two-way (server and client authentication) SSL are supported.

Procedure

1. Create a keystore file:

```
cd /usr/jdk64/jdk1.8.0_112/bin/
keytool -genkey -alias serverkey -keypass <keypass> -keyalg RSA -sigalg
 SHA1withRSA -keystore atlas.keystore -storepass <keypass> -validity 3650
 -dname "CN=Nicola Marangoni, OU=PS, O=Hortonworks, L=Munich, ST=BY, C=DE"
```

2. Create a .jceks file:

```
cd /usr/hdp/current/atlas-server/bin
./cputil.py
Please enter the full path to the credential provider:jceks://file/home/
atlas/test.jceks
0    [main] WARN  org.apache.hadoop.util.NativeCodeLoader  - Unable
 to load native-hadoop library for your platform... using builtin-java
 classes where applicable
Please enter the password value for keystore.password:<keypass>
Please enter the password value for keystore.password again:<keypass>
Please enter the password value for truststore.password:<keypass>
Please enter the password value for truststore.password again:<keypass>
Please enter the password value for password:<keypass>
Please enter the password value for password again:<keypass>
cp /root/atlas.keystore /home/atlas/
cd /home/atlas
```

3. Assign 440 rights to both of these files, and make atlas:hadoop owners for these files (so that Atlas can read these files):chmod 440 atlas.keystore test.jceks.

4. Select **Atlas** > **Configs** > **Advanced**, then select Advanced application-properties and set the following properties:

Table 1: Atlas Advanced application-properties

Property	Value	Description
atlas.enableTLS	true	Enable or disable the SSL listener. Set this value to true to enable SSL (default value is false).

Add the following properties by selecting Custom application-properties > Add Property.

Table 2: Atlas Custom application-properties

Property	Value	Description
keystore.file	/home/atlas/atlas.keystore	The path to the keystore file leveraged by the server. This file contains the server certificate.
truststore.file	/home/atlas/atlas.keystore	The path to the truststore file. This file contains the certificates of other trusted entities (e.g. the certificates for client processes if two-way SSL is enabled). In most instances this can be set to the same value as the keystore.file property (especially if one-way SSL is enabled).
client.auth.enabled	true	Enable/disable client authentication (disabled by default). If enabled, the client must authenticate to the server during the transport session key creation process (i.e. two-way SSL is in effect).
cert.stores.credential.provider.path	jceks://file//home/atlas/test.jceks	The path to the Credential Provider store file. The passwords for the keystore, truststore, and server certificate are maintained in this secure file. Utilize the cputil script in the 'bin' directory (see below) to populate this file with the passwords required.
atlas.ssl.exclude.cipher.suites	*NULL.*, .*RC4.*, .*MD5.*, .*DES.*, .*DSS.*	The excluded Cipher Suites list - NULL.,.*RC4.*,.*MD5.*,.*DES.*,.*DSS.* are weak and unsafe Cipher Suites that are excluded by default. If additional Ciphers need to be excluded, set this property with the default Cipher Suites such as *NULL.*, .*RC4.*, .*MD5.*, .*DES.*, .*DSS.*, and add the additional Cipher Suites to the list with a comma separator. They can be added with their full name or a regular expression. The Cipher Suites listed in the atlas.ssl.exclude.cipher.suites property take precedence over the default Cipher Suites. You should retain the default Cipher Suites, and add additional ones to increase security.

Note:

Enabling or disabling HTTPS will not automatically reconfigure the atlas.rest.address property. To update this property, select Atlas > Configs > Advanced, then select Advanced application-properties. Change the URL strings in the atlas.rest.address property to "https" if SSL is enabled (if the atlas.enableTLS property is set to true) "https". If SSL is not enabled, use "http". For example:

```
http:<server_one>:21000,http:<server_two>:21000,http:<server_three>:21000
https:<server_one>:21443,https:<server_two>:21443,https:<server_three>:21443
```

The default HTTP port is 21000 and the default HTTPS port is 21443. These values can be overridden using the atlas.server.http.port and atlas.server.https.port properties, respectively.

5. After manually editing these settings, select Actions > Stop All on the Ambari dashboard to stop all services, then select Actions > Start All to restart all services.

What to do next

Note:

If you disable Atlas SSL, you must clear your browser cookies in order to log in to the Atlas UI using HTTP request headers.

SSL for Apache Atlas Credential Provider Utility Script

How to create the credential provider for Atlas when enabling SSL for Atlas.

About this task

In order to prevent the use of clear-text passwords, the Atlas platform uses the Credential Provider facility for secure password storage (see the Hadoop Credential Command Reference for more information about this facility). The cputil script can be used to create the required password store.

To create the credential provider for Atlas:

Procedure

1. Switch to the Atlas bin directory: cd /usr/hdp/current/atlas-server/bin.
2. Run the following command: ./cputil.py.
3. When prompted, enter the path for the generated credential provider. The format for the path is: /local/file/path/file.jceks.

   Only one absolute path is allowed. The credential provider files generally use the .jceks extension.
4. When prompted, enter the passwords for the keystore, truststore, and server key (these passwords must match the passwords used when actually creating the associated certificate store files).
5. The credential provider is generated and saved to the specified path.

Related Information

Hadoop Commands Guide> credential

SPNEGO setup for WebHCat

How to set up SPNEGO for WebHCat.

Procedure

To set up secure WebHCat, set the following properties in the /etc/hcatalog/conf/webhcat-site.xml file:

```
</property>
    <name>templeton.kerberos.principal</name>
    <value>HTTP/host1234.example.com@EXAMPLE.COM</value>
    <description/>
<property>
```

The templeton.kerberos.principal property must use the host name of the WebHCat Server.

```
<property>
    <name>templeton.kerberos.keytab</name>
    <value>/etc/security/keytabs/spnego.service.keytab</value>
```

```
    <description/>
</property>
```

```
<property>
    <name>templeton.kerberos.secret</name>
    <value>secret</value>
    <description/>
</property>
```

```
<property>
    <name>templeton.hive.properties</name>
    <value>hive.metastore.local=false,hive.metastore.uris=thrift://
host1234.example.com:9083,

 hive.metastore.sasl.enabled=true,hive.metastore.execute.setugi=true,
                    hive.exec.mode.local.auto=false,
                    hive.metastore.kerberos.principal=hive/
_HOST@EXAMPLE.COM</value>
         <description>Properties to set when running hive.</description>
</property>
```

Be sure to set the templeton.hive.properties property with the host name for your Thrift server.

Configure SSL for Knox

For the simplest of evaluation deployments, the initial startup of the Knox Gateway will generate a self-signed cert for use on the same machine as the gateway instance. These certificates are issued for "localhost" and will require specifically disabling hostname verification on client machines other than where the gateway is running.

Create Self-Signed Certificate with Specific Hostname for Evaluations

How to create a self-signed certificate with a specific hostname for evaluations, when configuring SSL for Knox.

About this task

In order to continue to use self-signed certificates for larger evaluation deployments, a certificate can be generated for a specific hostname. This will allow clients to properly verify the hostname presented in the certificate as the host that they requested in the request URL.

Procedure

1. >Create a certificate: where $gateway-hostname is the FQDN of the Knox Gateway: cd $gateway bin/knoxcli.cmd create-cert --hostname $gateway-hostname.
2. Export the certificate in PEM format: keytool -export -alias gateway-identity -rfc -file $certificate_path -keystore $gateway /data/security/keystores/gateway.jks.

   Note:

   cURL option accepts certificates in PEM format only.
3. Restart the gateway: cd $gateway bin/gateway.sh stop bin/gateway.sh start.
4. After copying the certificate to a client, use the following command to verify: curl --cacert $certificate_path -u $username : $password https:// $gateway-hostname : $gateway_port /gateway/ $cluster_name /webhdfs/v1?op=GETHOMEDIRECTORY.

Create CA-Signed Certificates for Production

How to create a CA-signed certificate for production, when configuring SSL for Knox.

About this task

For production deployments or any deployment in which a certificate authority issued certificate is needed, the following steps are required.

Procedure

1. Import the desired certificate/key pair into a java keystore using keytool and ensure the following:
   - The certificate alias is gateway-identity.
   - The store password matches the master secret created earlier.
   - Note the key password used - as we need to create an alias for this password.
2. Add a password alias for the key password:cd $gateway bin/knoxcli.cmd create-cert create-alias gateway-identity-passphrase --value $actualpassphrase.

Note:

The password alias must be gateway-identity-passphrase.

Set Up Trust for the Knox Gateway Clients

How to set up trust for the Knox Gateway clients, when configuring SSL for Knox.

About this task

In order for clients to trust the certificates presented to them by the gateway, they will need to be present in the client's truststore as follows.

Procedure

1. Export the gateway-identity cert from the $gateway /data/security/keystores/gateway.jks using java keytool or another key management tool.
2. Add the exported certificate to the cacerts or other client specific truststore or the gateway.jks file can be copied to the clients to be used as the truststore.

Note:

If taking this approach be sure to change the password of the copy so that it no longer matches the master secret used to protect server side artifacts.

Securing Phoenix

To configure Phoenix to run in a secure Hadoop cluster, use the instructions on “Configuring Phoenix to Run in a Secure Cluster”.

Set Up SSL for Ambari

How to set up SSL for Ambari.

About this task

If you want to limit access to the Ambari Server to HTTPS connections, you need to provide a certificate. While it is possible to use a self-signed certificate for initial trials, they are not suitable for production environments. After your certificate is in place, you must run a special setup command.

Ambari Server should not be running when you do this. Either make these changes before you start Ambari the first time, or bring the server down before running the setup command.

Procedure

1. Log into the Ambari Server host.
2. Locate your certificate. If you want to create a temporary self-signed certificate, use this as an example:

```
openssl genrsa -out $wserver.key 2048
openssl req -new -key $wserver.key -out $wserver.csr
openssl x509 -req -days 365 -in $wserver.csr -signkey $wserver.key -out
 $wserver.crt
```

   Where $wserver is the Ambari Server host name.

   The certificate you use must be PEM-encoded, not DER-encoded. If you attempt to use a DER-encoded certificate, you see the following error:

   unable to load certificate 140109766494024:error:0906D06C:PEM routines:PEM_read_bio:no start line:pem_lib.c :698:Expecting: TRUSTED CERTIFICATE

   You can convert a DER-encoded certificate to a PEM-encoded certificate using the following command:

   openssl x509 -in cert.crt -inform der -outform pem -out cert.pem

   where cert.crt is the DER-encoded certificate and cert.pem is the resulting PEM-encoded certificate.
3. Run the special setup command and answer the prompts: ambari-server setup-security.
   a) Select 1 for Enable HTTPS for Ambari server.
   b) Respond y to Do you want to configure HTTPS ?.
   c) Select the port you want to use for SSL. The default port number is 8443.
   d) Provide the complete path to your certificate file ($wserver.crt from above) and private key file ($wserver.key from above).
   e) Provide the password for the private key.
   f) Start or restart the Server: ambari-server restart.
4. Trust Store Setup - If you plan to use Ambari Views with your Ambari Server, after enabling SSL for Ambari using the instructions below, you must also “Set Up Truststore for Ambari Server”.

Related Information

Set Up Truststore for Ambari Server

Set Up Truststore for Ambari Server

If you plan to set up SSL for Ambari or to enable wire encryption for HDP, you must configure the Truststore for Ambari and add certificates.

About this task

Ambari Server should not be running when you do this. Either make these changes before you start Ambari the first time, or bring the server down before running the setup command.

Procedure

1. On the Ambari Server, create a new keystore that will contain the Ambari Server's HTTPS certificate:
   a) keytool -import -file <path_to_the_Ambari_Server's_SSL_Certificate> -alias ambari-server -keystore ambari-server-truststore.
   b) When prompted to 'Trust this certificate?' type "yes".
2. Configure the ambari-server to use this new trust store:

```
ambari-server setup-security
Using python  /usr/bin/python2.6
Security setup options...
===========================================================================
Choose one of the following options:
```

```
  [1] Enable HTTPS for Ambari server.
  [2] Encrypt passwords stored in ambari.properties file.
  [3] Setup Ambari kerberos JAAS configuration.
  [4] Setup truststore.
  [5] Import certificate to truststore.
==========================================================================
Enter choice, (1-5): *4*
Do you want to configure a truststore [y/n] (y)? *y*
TrustStore type [jks/jceks/pkcs12] (jks): *jks*
Path to TrustStore file : *<path to the ambari-server-truststore
 keystore>*
Password for TrustStore:
Re-enter password:
Ambari Server 'setup-security' completed successfully.
```

3. Once configured, the Ambari Server must be restarted for the change to take effect: ambari-server restart.

Related Information

Set Up SSL for Ambari

Set Up Two-Way SSL Between Ambari Server and Ambari Agents

Two-way SSL provides a way to encrypt communication between Ambari Server and Ambari Agents. By default Ambari ships with Two-way SSL disabled. To enable Two-way SSL:

Before you begin

Ambari Server should not be running when you do this: either make the edits before you start Ambari Server the first time or bring the server down to make the edits.

Procedure

1. On the Ambari Server host, open /etc/ambari-server/conf/ambari.properties with a text editor.
2. Add the following property: security.server.two_way_ssl = true.
3. Start or restart the Ambari Server: ambari-server restart.

What to do next

The Agent certificates are downloaded automatically during Agent Registration.

Optional: Recreating the Ambari SSL Certificate Authority

The Ambari Certificate Authority, (CA) issues digital certificates used to securely connect Ambari Server and Ambari Agents. An Ambari CA certificate is valid for 365 days. If an Ambari CA certificate is corrupted, or has expired, you must recreate the CA, causing a new CA certificate and new SSL certificates for each Ambari agent to be created. This solution assumes that the certificates signed by the Ambari CA are replaceable - which is generally the case for certificates used by Ambari agents for two-way SSL connections.

Procedure

On the Ambari server:

1. Stop the Ambari server: ambari-server stop.
2. Backup /var/lib/ambari-server/keys and its child directories.
3. Delete the following files from /var/lib/ambari-server/keys:
   - ca.key
   - ca.csr
   - ca.crt
   - pass.txt

- keystore.p12
- *.csr
- *.crt

4. Delete the following files from /var/lib/ambari-server/keys/db:
   - index.txt.old
   - index.txt.attr.old
   - serial.old
5. Truncate the following files from /var/lib/ambari-server/keys/db:
   - index.txt
   - index.txt.attr
6. Edit the following file from /var/lib/ambari-server/keys/db:

   In serial, set the contents to be exactly:

   ```
   00
   ```
7. Delete all files under /var/lib/ambari-server/keys/db/newcerts.
8. Restart Ambari server: ambari-server restart.
   After restarting the Ambari server, the following (or similar) entries should be seen in the /var/log/ambari-server/ambari-server.log file:

   ```
   12 Jun 2017 14:38:19,606  INFO [main] ShellCommandUtil:63 - Command
    openssl genrsa -des3 -passout pass:**** -out /var/lib/ambari-server/keys/
   ca.key 4096  was finished with exit code: 0 - the operation was completely
    successfully
   12 Jun 2017 14:38:19,640  INFO [main] ShellCommandUtil:63 - Command
    openssl req -passin pass:**** -new -key /var/lib/ambari-server/keys/
   ca.key -out /var/lib/ambari-server/keys/ca.csr -batch was finished with
    exit code: 0 - the o
   peration was completely successfully.
   12 Jun 2017 14:38:19,683  INFO [main] ShellCommandUtil:63 - Command
    openssl ca -create_serial -out /var/lib/ambari-server/keys/ca.crt -days
    365 -keyfile /var/lib/ambari-server/keys/ca.key -key **** -selfsign -
   extensions jdk7_ca -config /var/lib/ambari-server/keys/ca.config -batch -
   infiles /var/lib/ambari-server/keys/ca.csr was finished with exit code: 0
    - the operation was completely successfully.
   12 Jun 2017 14:38:19,701  INFO [main] ShellCommandUtil:63 - Command
    openssl pkcs12 -export -in /var/lib/ambari-server/keys/ca.crt -inkey /
   var/lib/ambari-server/keys/ca.key -certfile /var/lib/ambari-server/keys/
   ca.crt -out /var/lib/ambari-server/keys/keystore.p12 -password pass:**** -
   passin pass:****
    was finished with exit code: 0 - the operation was completely
    successfully.
   12 Jun 2017 14:38:19,708  INFO [main] ShellCommandUtil:63 - Command find /
   var/lib/ambari-server/keys -type f -exec chmod 700 {} + was finished with
    exit code: 0 - the operation was completely successfully.
   12 Jun 2017 14:38:19,708  INFO [main] ShellCommandUtil:63 - Command chmod
    600 /var/lib/ambari-server/keys/pass.txt was finished with exit code: 0 -
    the operation was completely successfully.
   12 Jun 2017 14:52:53,797  INFO [qtp-ambari-agent-34]
    CertificateManager:200 - Signing agent certificate
   12 Jun 2017 14:52:53,800  INFO [qtp-ambari-agent-34]
    CertificateManager:220 - Validating agent hostname:
    c6401.ambari.apache.org
   12 Jun 2017 14:52:53,800  INFO [qtp-ambari-agent-34]
    CertificateManager:232 - Verifying passphrase
   12 Jun 2017 14:52:53,849  INFO [qtp-ambari-agent-34] ShellCommandUtil:63
    - Command openssl ca -config /var/lib/ambari-server/keys/ca.config -
   in /var/lib/ambari-server/keys/c6401.ambari.apache.org.csr -out /var/lib/
   ```

```
ambari-server/keys/c6401.ambari.apache.org.crt -batch -passin pass:**** -
keyfile /var/lib/ambari-server/keys/ca.key -cert /var/lib/ambari-server/
keys/ca.crt was finished with exit code: 0 - the operation was completely
 successfully.
```

On each Ambari agent host:

9. Stop the Ambari agent: ambari-agent stop.
10. Backup /var/lib/ambari-agent/keys and its child directories.
11. Delete the following files from /var/lib/ambari-server/keys:
    - ca.crt
    - *.crt
    - *.csr
    - *.key
12. Restart Ambari agent: ambari-agent restart.
    After restarting the Ambari agent, the following (or similar) entries should be seen in the /var/log/ambari-agent/ambari-agent.log file:

```
INFO 2017-06-12 14:52:53,625 security.py:55 - Server require two-way SSL
 authentication. Use it instead of one-way...
INFO 2017-06-12 14:52:53,625 security.py:179 - Server certicate not
 exists, downloading
INFO 2017-06-12 14:52:53,625 security.py:202 - Downloading server cert
 from https://localhost:8440/cert/ca/
INFO 2017-06-12 14:52:53,693 security.py:187 - Agent key not exists,
 generating request
INFO 2017-06-12 14:52:53,693 security.py:258 - openssl req -new
 -newkey rsa:1024 -nodes -keyout "/var/lib/ambari-agent/keys/
c6401.ambari.apache.org.key" -subj /OU=c6401.ambari.apache.org/ -out "/
var/lib/ambari-agent/keys/c6401.ambari.apache.org.csr"
INFO 2017-06-12 14:52:53,736 security.py:195 - Agent certificate not
 exists, sending sign request
INFO 2017-06-12 14:52:53,855 security.py:93 - SSL Connect being called..
 connecting to the server
INFO 2017-06-12 14:52:53,933 security.py:77 - SSL connection established.
 Two-way SSL authentication completed successfully.
```

Configure Ranger SSL

How to configure Ranger SSL in an Ambari-enabled cluster.

Related Information

Configure Non-Ambari Ranger SSL

Configure Non-Ambari Ranger SSL

Configuring Public CA Certificates (Ranger SSL)

How to configure Ranger SSL using public CA Certificates, when configuring Ranger SSL in an Ambari cluster.

If you have access to Public CA issued certificates, use the following steps to configure Ambari Ranger SSL.

Prerequisites

Prerequisites for setting up Ambari Ranger SSL using Public CA certificates.

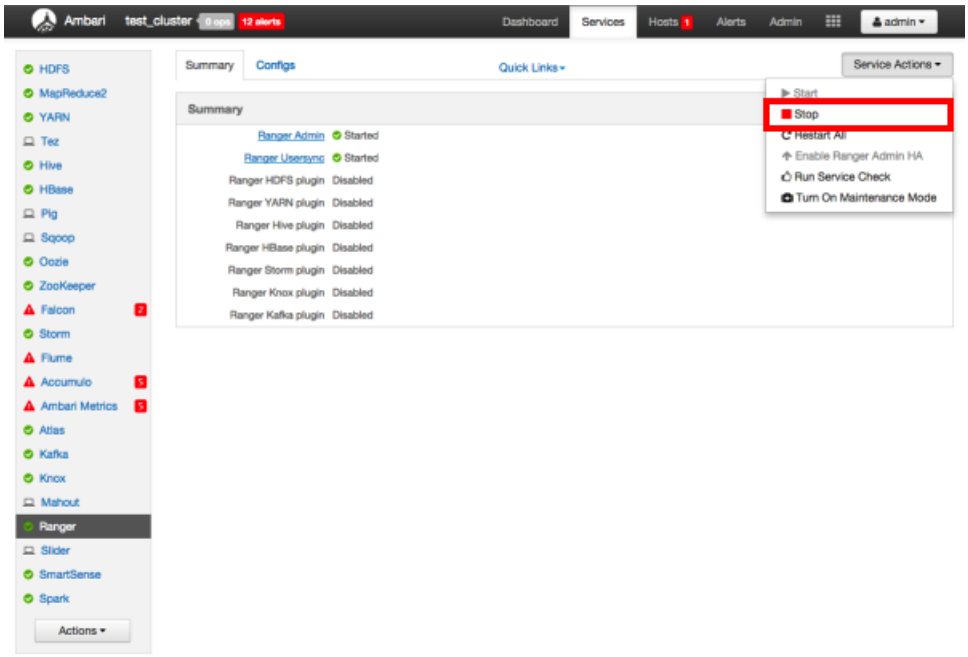
- Copy the keystore/truststore files into a different location (e.g. /etc/security/serverKeys) than the /etc/<component>/conf folder.
- Make sure that the JKS file names are unique.
- Make sure that the correct permissions are applied.
- Make sure that passwords are secured.

Configure Ranger Admin

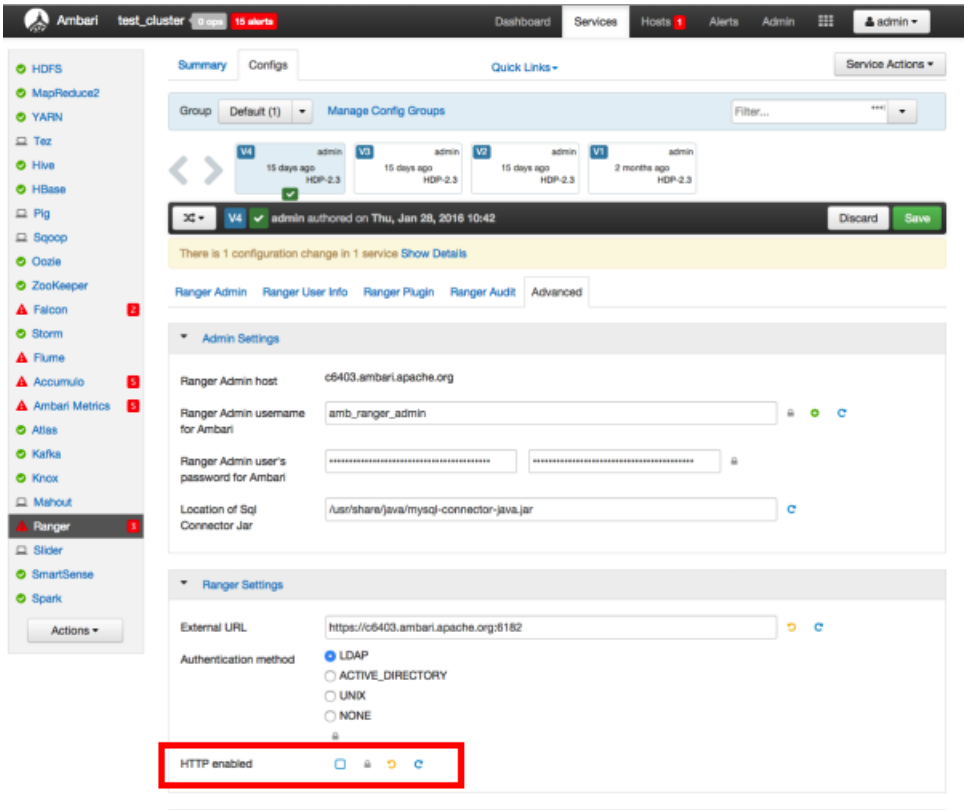
How to configure Ranger Admin, when setting up Ambari Ranger SSL using Public CA certificates.

Procedure

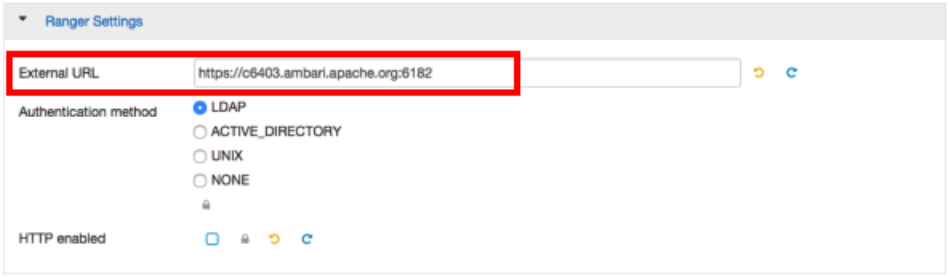
1. Stop Ranger by selecting **Ranger** > **Service Actions** > **Stop**.



2. Disable the HTTP port and enable the HTTPS port with the required keystore information.
   a) Select Configs > Advanced. Under Ranger Settings, clear the HTTP enabled check box (this blocks all agent calls to the HTTP port even if the port is up and working).

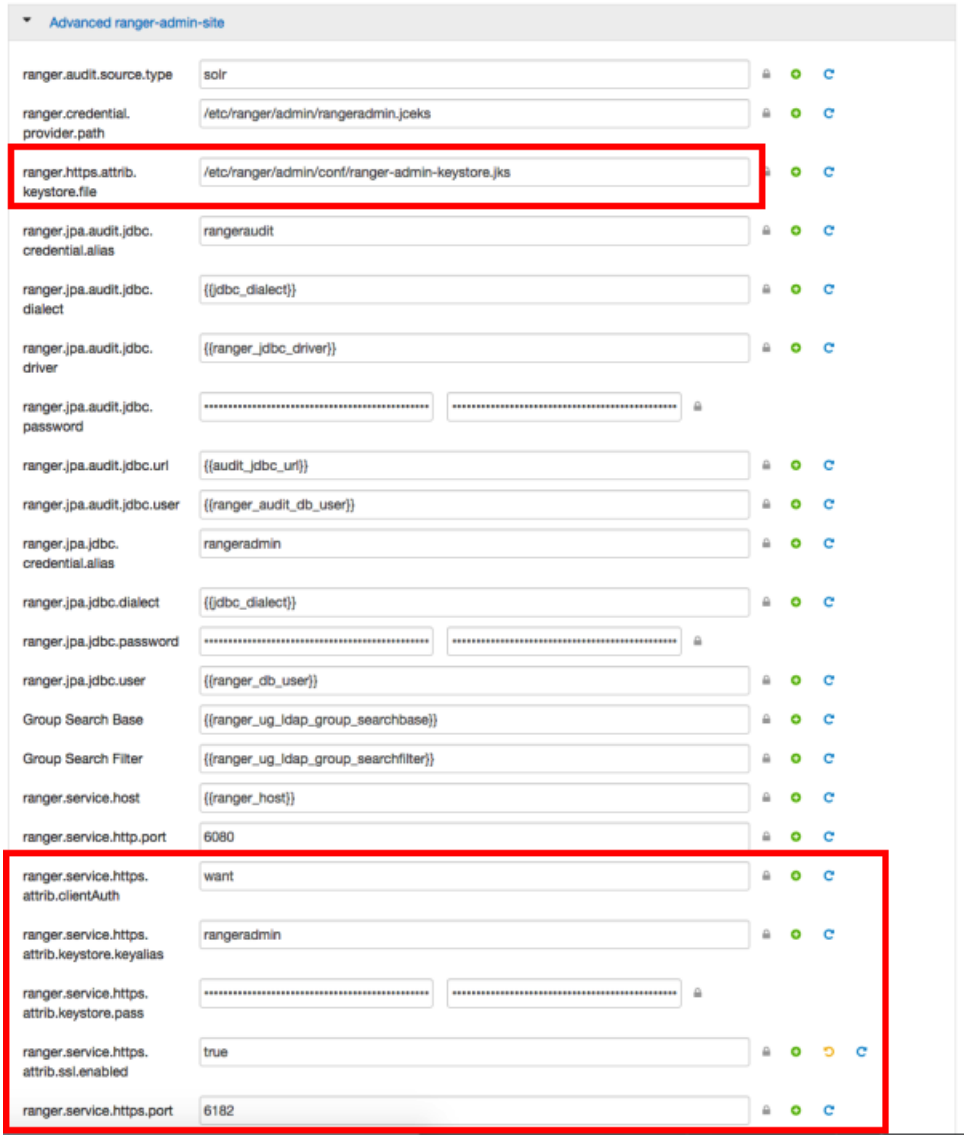


b) Under Ranger Settings, provide the value in the External URL box in the format https://<hostname of policy manager>:<https port>.

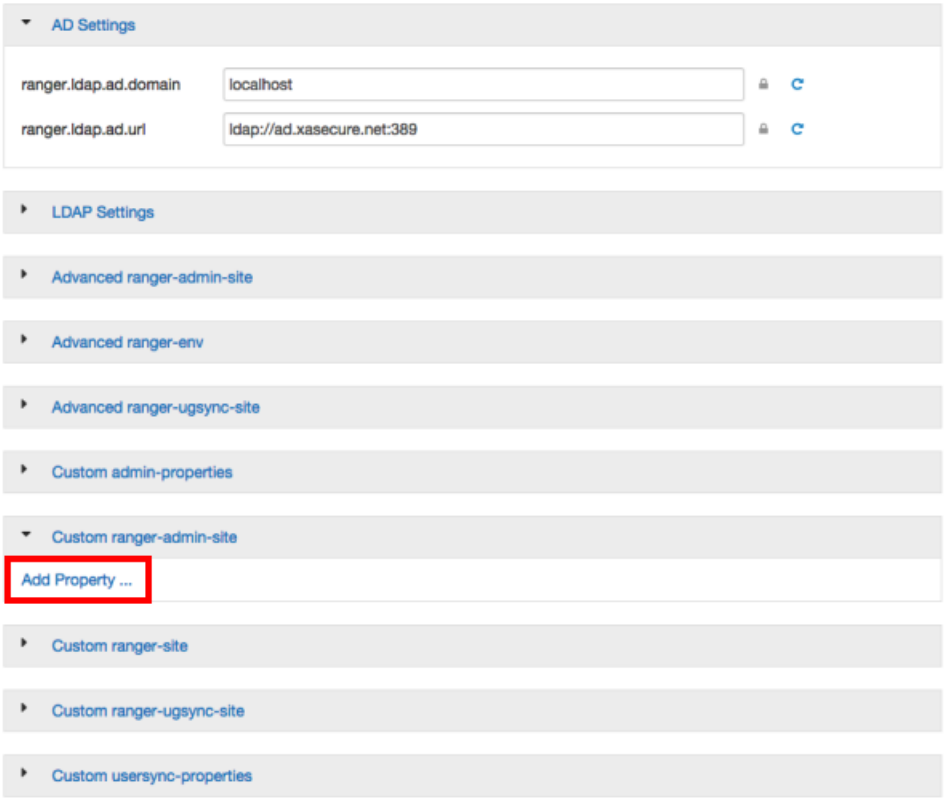


c) Under Advanced ranger-admin-site, set the following properties:

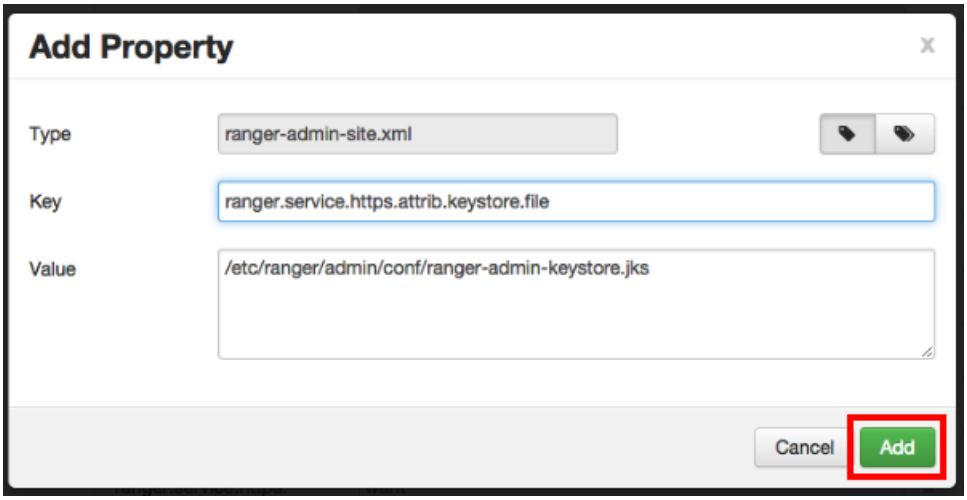
- ranger.https.attrib.keystore.file -- Provide the location of the Public CA issued keystore file.
- ranger.service.https.attrib.keystore.pass -- Enter the password for the keystore.
- ranger.service.https.attrib.keystore.keyalias -- Enter the alias name for the keystore private key.
- ranger.service.https.attrib.clientAuth -- Enter want as the value. This validates the client cert from all agents, but not the requests from web applications. Setting this value to want requires the client to have a certificate to use to sign traffic. If you do not want to put certificates on the client machines to do two-way SSL, this parameter can be set to false to enable one-way SSL.
- ranger.service.https.attrib.ssl.enabled -- set this property to true.
- ranger.service.https.port -- Make sure that this port is available, or change the value to an available port number.



3. Under Custom ranger-admin-site, add the following properties:
   - ranger.service.https.attrib.keystore.file -- Specify the same value provided for the ranger.https.attrib.keystore.file property.
   - ranger.service.https.attrib.client.auth -- Specify the same value provided for the ranger.service.https.attrib.clientAuth property.
4. To add a Custom ranger-admin-site property:
   a) Select Custom ranger-admin-site, then click Add Property.



b) On the Add Property pop-up, type the property name in the Key box, type the property value in the Value box, then click Add.



5. Save your changes and start Ranger Admin.

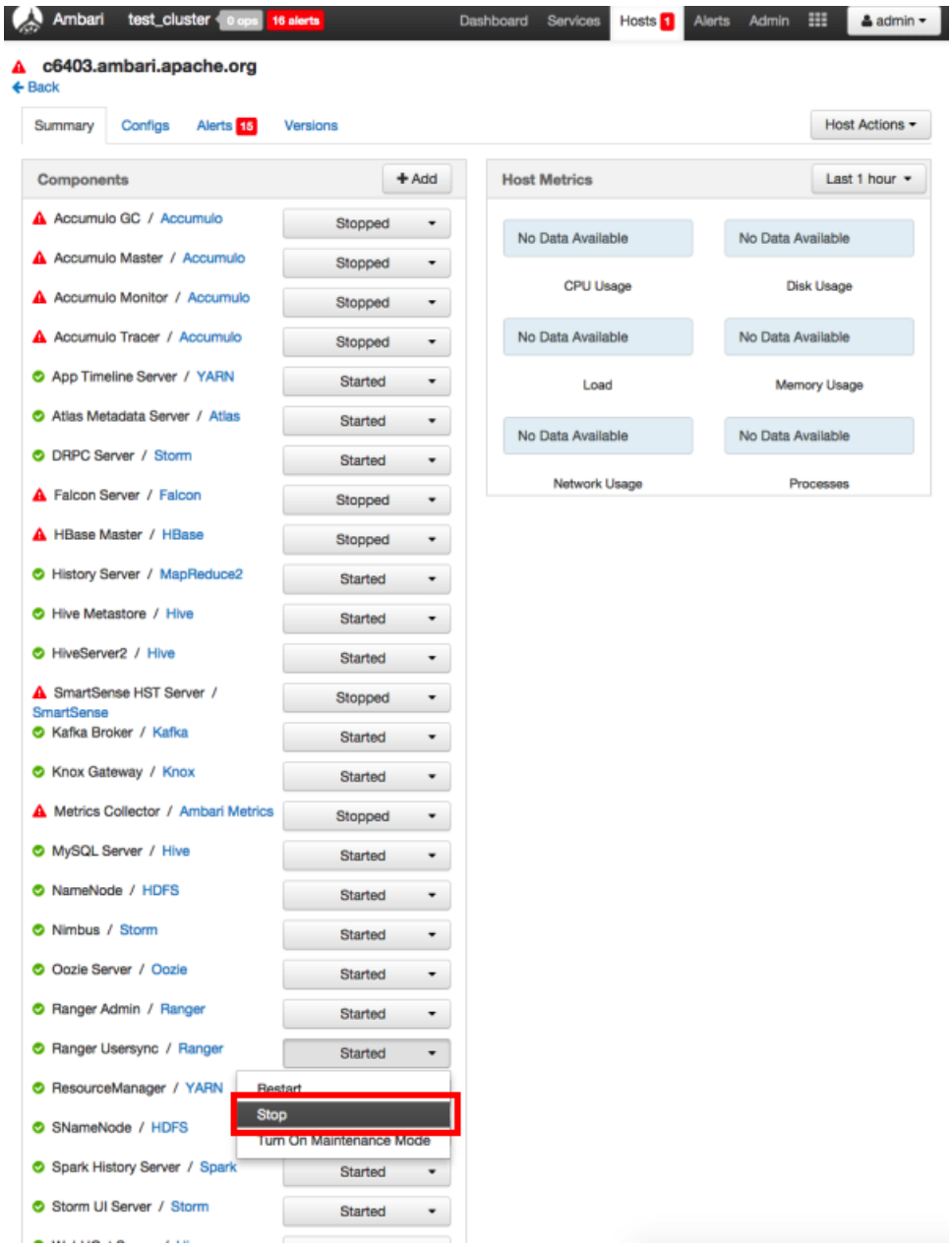
   When you attempt to access the Ranger Admin UI with the HTTPS protocol on the port specified by the ranger.service.https.port property, the browser should report that it does not trust the site. Click Proceed anyway and you should be able to access the Ranger Admin UI over HTTPS.

Configure Ranger Usersync

How to configure Ranger Usersync, when setting up Ambari Ranger SSL using Public CA certificates.

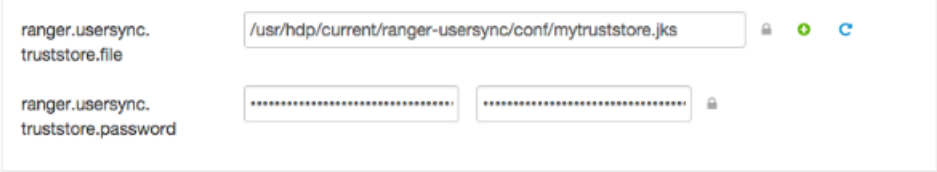
Procedure

1. Stop Ranger Usersync by selecting the Ranger Usersync link, then select Started > Stop next to Ranger Usersync.



2. Navigate back to Ranger and select Configs > Advanced, then click Advanced ranger-ugsync-site. Set the following properties:

   - ranger.usersync.truststore.file -- Enter the path to the truststore file.
   - ranger.usersync.truststore.password -- Enter the truststore password.



3. Start Ranger Usersync by selecting the Ranger Usersync link on the Summary tab, then select Stopped > Start next to Ranger Usersync.

Configuring Ranger Plugins for SSL (Public CA Certificates)

How to configure Ranger Plugins for SSL, when setting up Ambari Ranger SSL using Public CA certificates.

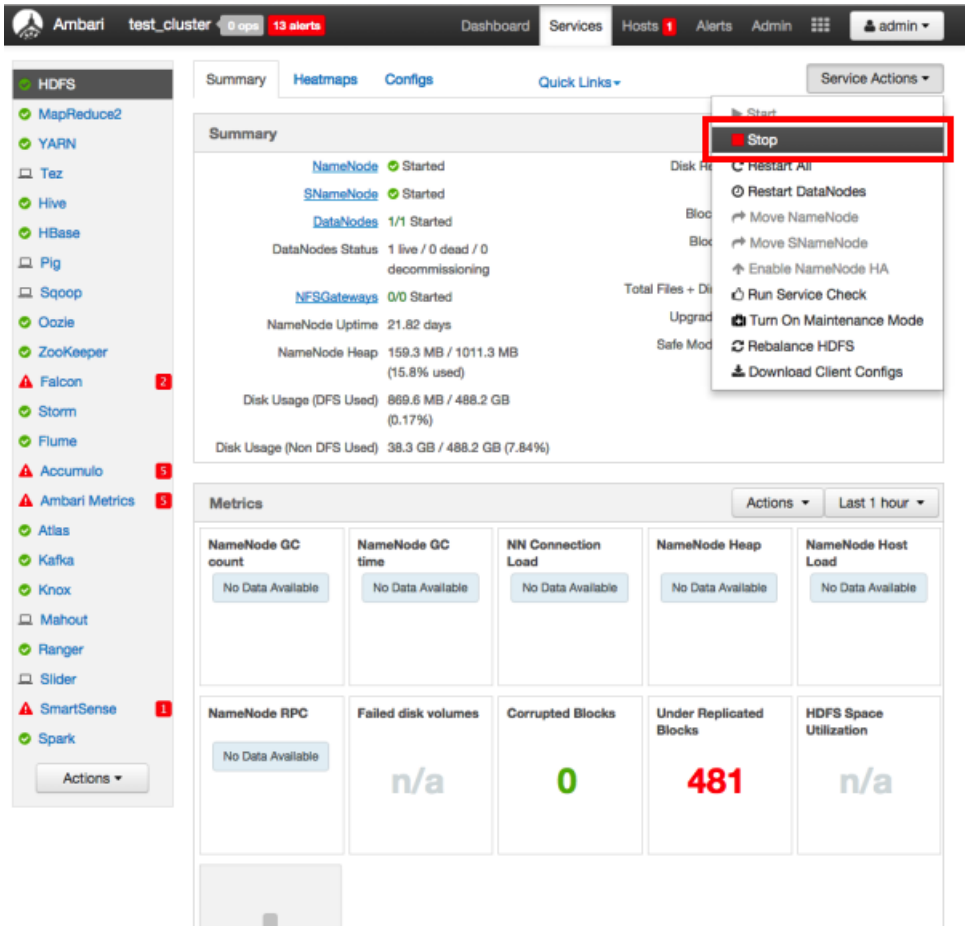
The following section shows how to configure the Ranger HDFS plugin for SSL. You can use the same procedure for other Ranger components. Additional steps required to configure the Ranger KMS plugin and server are provided in subsequent sections.

Configure the Ranger HDFS Plugin for SSL

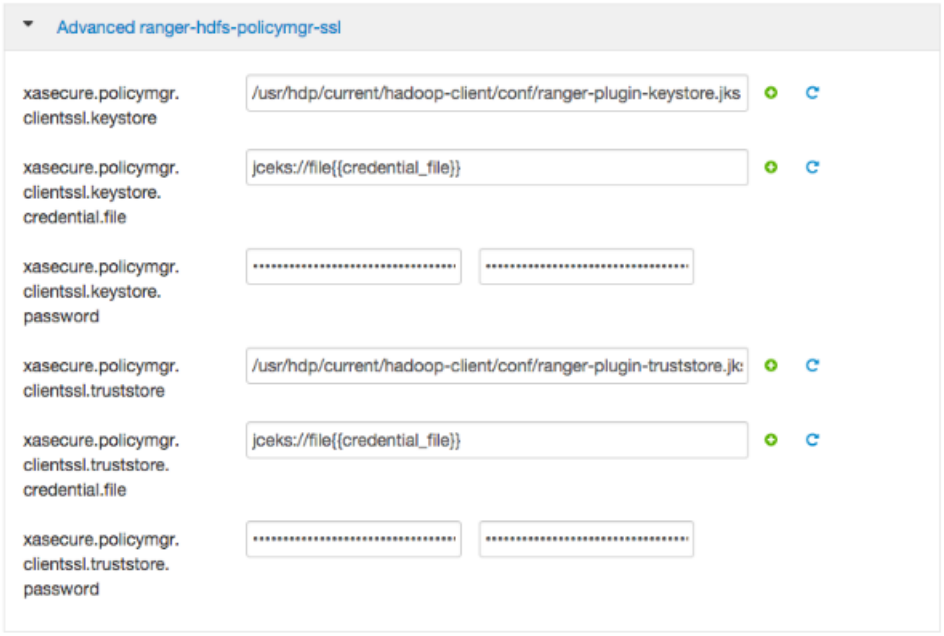
How to configure the Ranger HDFS Plugin for SSL, when setting up Ambari Ranger SSL using Public CA certificates. The following steps show how to configure the Ranger HDFS plugin for SSL. You can use the same procedure for other Ranger components.

Procedure

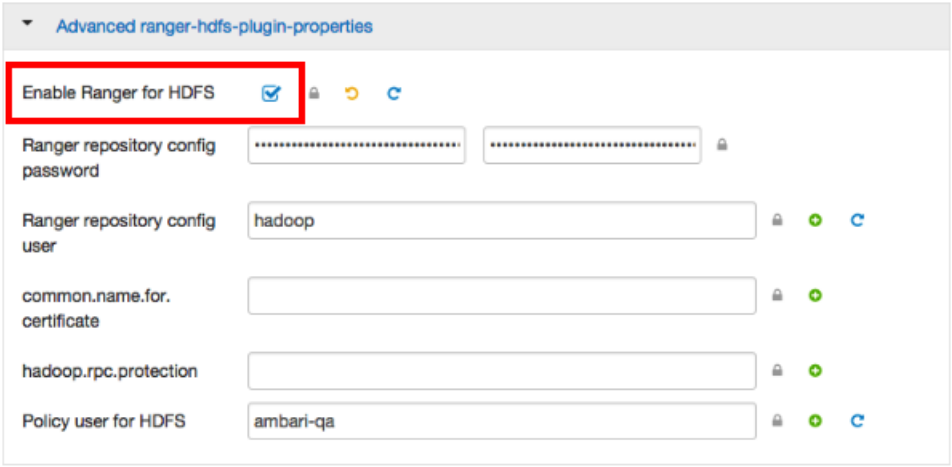
1. Stop HDFS by selecting **HDFS** > **Service Actions** > **Stop**.



2. Under **Ranger** > **Configs** > **Advanced** > **Ranger Settings**, provide the value in the External URL box in the format https://<hostname of policy manager>:<https port>.
3. Under **HDFS** > **Confgs** > **Advanced**, select Advanced ranger-hdfs-policymgr-ssl and set the following properties:
   - xasecure.policymgr.clientssl.keystore -- Enter the public CA signed keystore for the machine that is running the HDFS agent.
   - xasecure.policymgr.clientssl.keystore.password -- Enter the keystore password.



4. Select Advanced ranger-hdfs-plugin-properties, then select the Enable Ranger for HDFS check box.



5. Click Save at the top.
6. Start HDFS by selecting **HDFS** > **Service Actions** > **Start**.
7. Restart Ranger Admin: **Hosts** > **<Select host>** > **Ranger Admin / Ranger**, from the drop-down menu, select **Restart**.

   Or: service ranger-admin restart
8. Log into the Ranger Policy Manager UI as the admin user. Click the Edit button of the HDFS repository and provide the CN name of the keystore as the value for Common Name For Certificate, then save your changes.
9. Start the HDFS service by selecting **HDFS** > **Service Actions** > **Start**.
10. Select Audit > Agents. You should see an entry for your repo name with HTTP Response Code 200.

Related Information

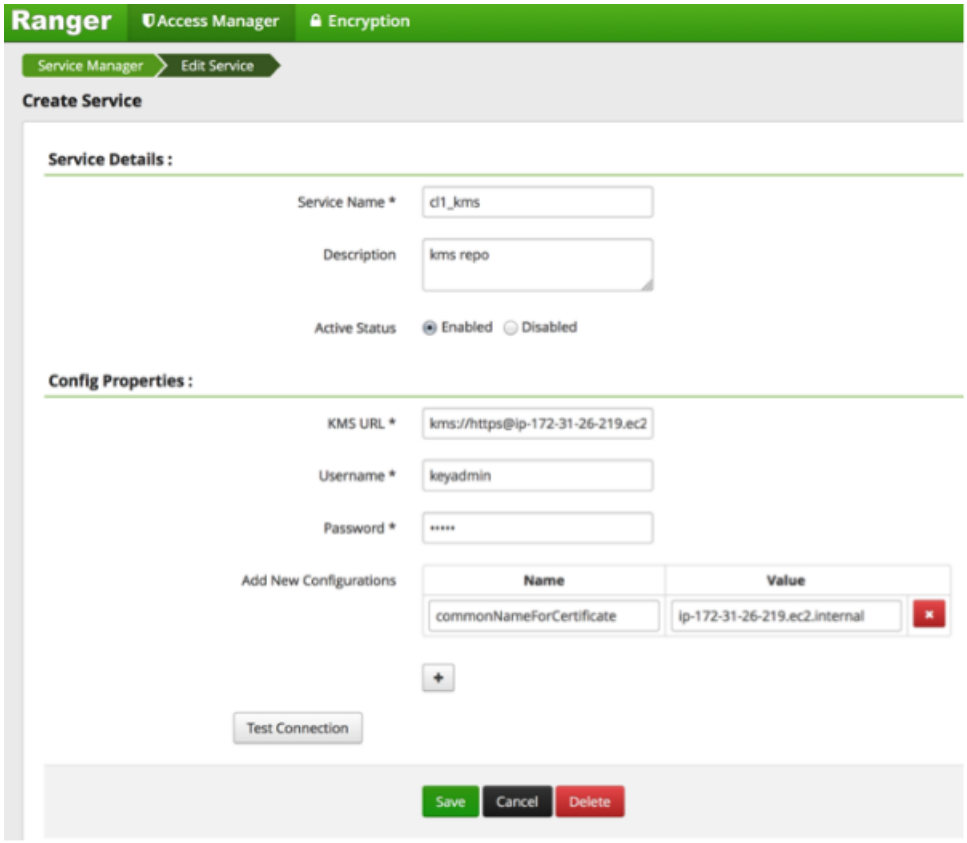
Configure the Ranger KMS Plugin for SSL
Configure Ranger HBase Plugin for SSL
Configure the Ranger KMS Plugin for SSL

Configure the Ranger KMS Plugin for SSL

How to configure the Ranger KMS Plugin for SSL, when setting up Ambari Ranger SSL using Public CA certificates. To configure the Ranger KMS (Key Management Service) plugin for SSL, use the procedure described in the task above, and then perform the following additional step.

Procedure

1. Complete “Configure the Ranger HDFS Plugin for SSL” (link below).
2. Log into the Policy Manager UI (as the keyadmin user) and click the Edit button of your KMS repository. Provide the CN name of the keystore as the value for Common Name For Certificate and save your changes. This property is not provided by default, so it must be added.



Related Information

Configure the Ranger HDFS Plugin for SSL

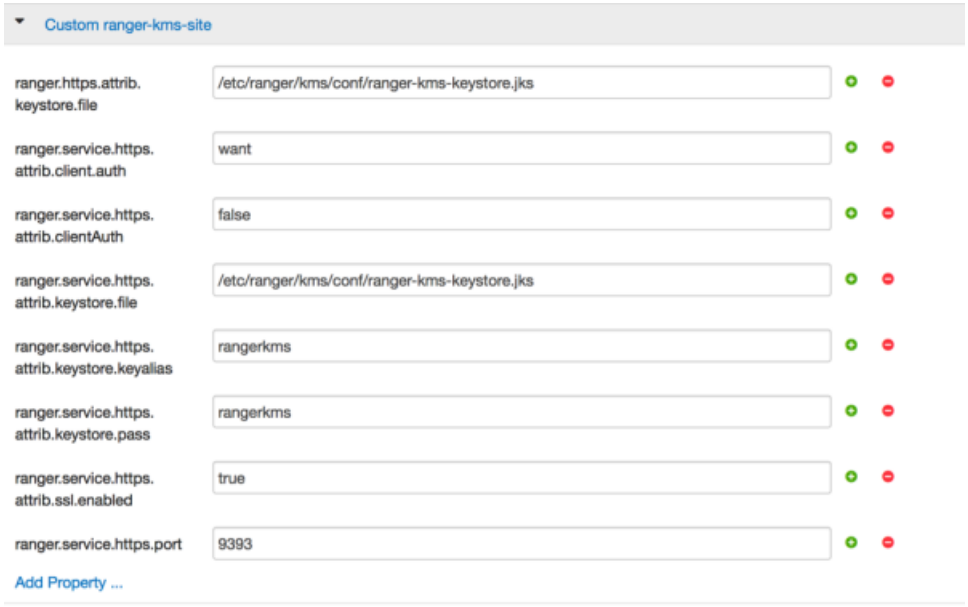
Configure the Ranger KMS Server for SSL

How to configure the Ranger KMS Server for SSL, when setting up Ambari Ranger SSL using Public CA certificates.

Procedure

1. Stop Ranger KMS by selecting Service Actions > Stop.
2. Select Custom ranger-kms-site, then add the following properties as shown below:
   - ranger.https.attrib.keystore.file
   - ranger.service.https.attrib.keystore.file (duplicate of above – workaround for now)
   - ranger.service.https.attrib.clientAuth
   - ranger.service.https.attrib.client.auth (duplicate of above – workaround for now)
   - ranger.service.https.attrib.keystore.keyalias
   - ranger.service.https.attrib.keystore.pass

- ranger.service.https.attrib.ssl.enabled
- ranger.service.https.port



3. Under Advanced kms_env, update the value of kms_port to match the value of ranger.service.https.port.
4. Save your changes and restart Ranger KMS.
   When you attempt to access the Ranger KMS UI with the HTTPS protocol on the port specified by the ranger.service.https.port property, the browser should report that it does not trust the site. Click Proceed anyway and you should be able to access the Ranger Admin UI over HTTPS.

Configure Ranger KMS Database for SSL-enabled MySQL

When an SSL-enabled database is configured for use with Ranger KMS, you must add certain configurations to Ranger. This explains how to configure the Ranger KMS Database for SSL-enabled MySQL, when setting up Ambari Ranger SSL using Public CA certificates.

Procedure

1. In Ambari>Ranger KMS>Configs>Advanced>Custom kms-properties, add the following parameters:
   - db_ssl_enabled=True
   - db_ssl_required=True
   - db_ssl_verifyServerCertificate=True
   - javax_net_ssl_keyStore=/etc/ranger/admin/keystore
   - javax_net_ssl_keyStorePassword=ranger
   - javax_net_ssl_trustStore=/etc/ranger/admin/truststore
   - javax_net_ssl_trustStorePassword=ranger

   Change keystore and truststore file paths according to your environment.

   If certificate verification is not required, you can set value false in property db_ssl_verifyServerCertificate. In this case, keystore and truststore file location need not to be valid and/or mandatory.
2. In Ambari>Ranger KMS>Configs>Advanced>Custom dbks-site, add the following parameters:
   - ranger.ks.db.ssl.enabled=true
   - ranger.ks.db.ssl.required=true
   - ranger.ks.db.ssl.verifyServerCertificate=true
   - ranger.ks.keystore.file=/etc/ranger/admin/keystore
   - ranger.ks.keystore.password=ranger

- ranger.ks.truststore.file=/etc/ranger/admin/truststore
- ranger.ks.truststore.password=password

Change keystore file path according to your environment.

If certificate verification is not required, then you can set value false in property ranger.db.ssl.verifyServerCertificate. In this case, keystore and truststore file location need not to be valid and/or mandatory.

3. Install/restart Ranger KMS.

Configure Ranger HBase Plugin for SSL

How to configure the Ranger HBase Plugin for SSL, when setting up Ambari Ranger SSL using Public CA certificates.

Procedure

Copy the truststore and keystore from the HBase master to all worker nodes running the RegionServers:

a) Complete “Configure the Ranger HDFS Plugin for SSL” (link below), modified for HBase.
b) From Ambari>HDFS>Configs>Advanced>Advanced ranger-hdfs-policymgr-ssl, copy the /path/keystore.file.name from xasecure.policymgr.clientssl.keystore and distribute it to all nodes.
c) From Ambari>HDFS>Configs>Advanced>Advanced ranger-hdfs-policymgr-ssl, copy the /path/truststore.file.name fromxasecure.policymgr.clientssl.truststore and distribute it to all nodes.
d) Restart HBase.

Related Information

Configure the Ranger HDFS Plugin for SSL

Configuring a Self-Signed Certificate (Ranger SSL)

How to configure Ranger SSL using self-signed certificates, when configuring Ranger SSL in an Ambari cluster.

If you do not have access to Public CA issued certificates, use the following steps to create a self-signed certificate and configure Ambari Ranger SSL.

Prerequisites

Prerequisites for setting up Ambari Ranger SSL using self-signed certificates.

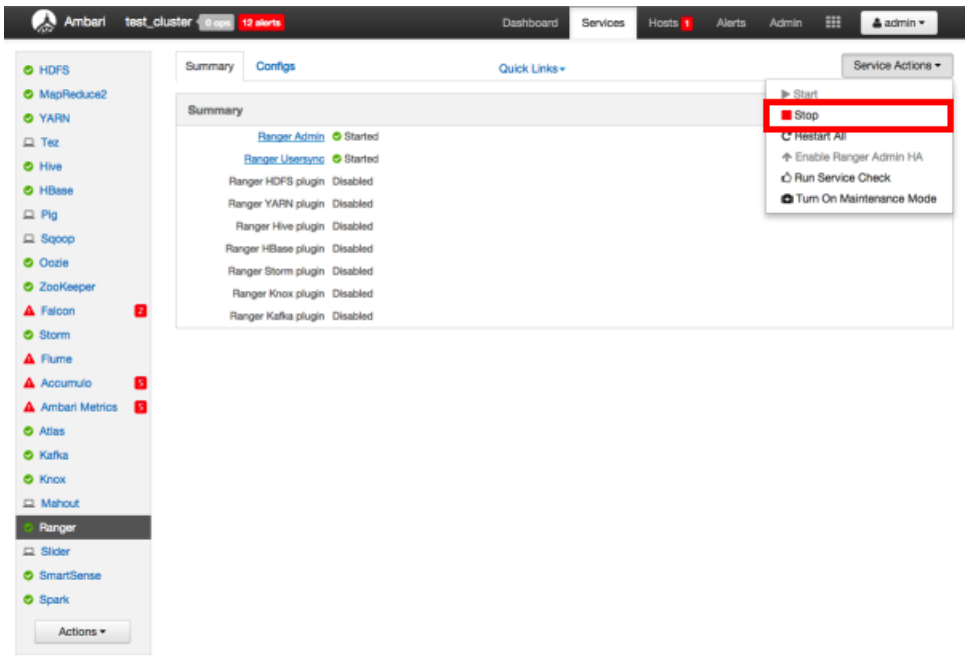
- Copy the keystore/truststore files into a different location (e.g. /etc/security/serverKeys) than the /etc/<component>/conf folder.
- Make sure that the JKS file names are unique.
- Make sure that the correct permissions are applied.
- Make sure that passwords are secured.

Configure Ranger Admin

How to configure Ranger Admin, when setting up Ambari Ranger SSL using self-signed certificates.

Procedure

1. Stop Ranger by selecting **Ranger** > **Service Actions** > **Stop**.

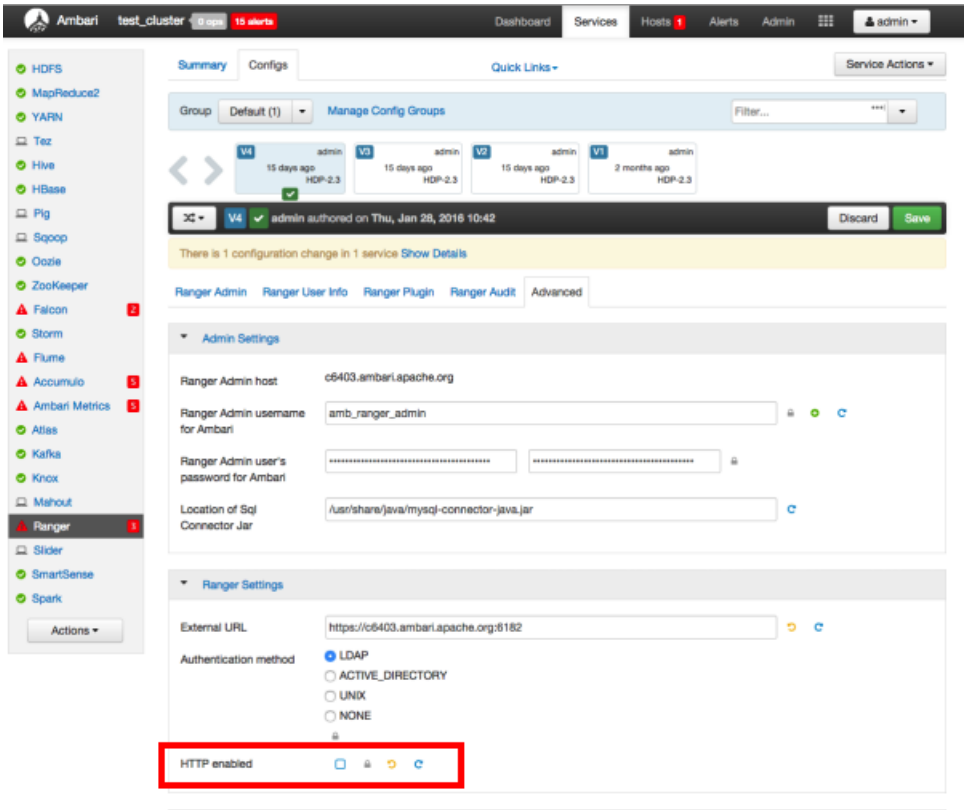


2. Change to the Ranger Admin directory and create a self-signed certificate.

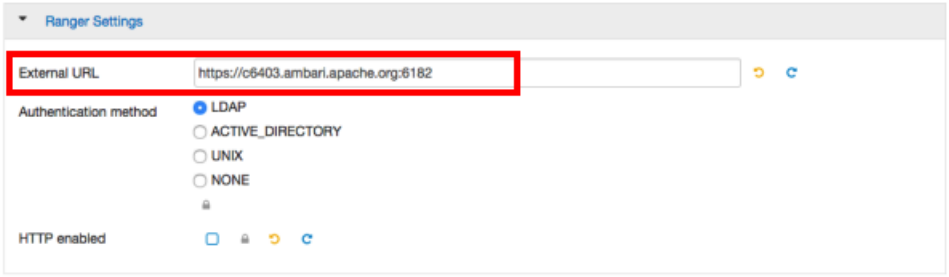
```
cd /etc/ranger/admin/conf
keytool -genkey -keyalg RSA -alias rangeradmin -keystore ranger-admin-
keystore.jks -storepass xasecure -validity 360 -keysize 2048
chown ranger:ranger ranger-admin-keystore.jks
chmod 400 ranger-admin-keystore.jks
```

   a) When prompted, provide the host name as the value for the "What is your first and last name?" question. then provide answers to the subsequent questions to create the keystore.
   b) When prompted for your password, press the Enter key. This will not work for Java keytool version 1.5.

3. Disable the HTTP port and enable the HTTPS port with the required keystore information.
   a) Select Configs > Advanced. Under Ranger Settings, clear the HTTP enabled check box (this blocks all agent calls to the HTTP port even if the port is up and working).

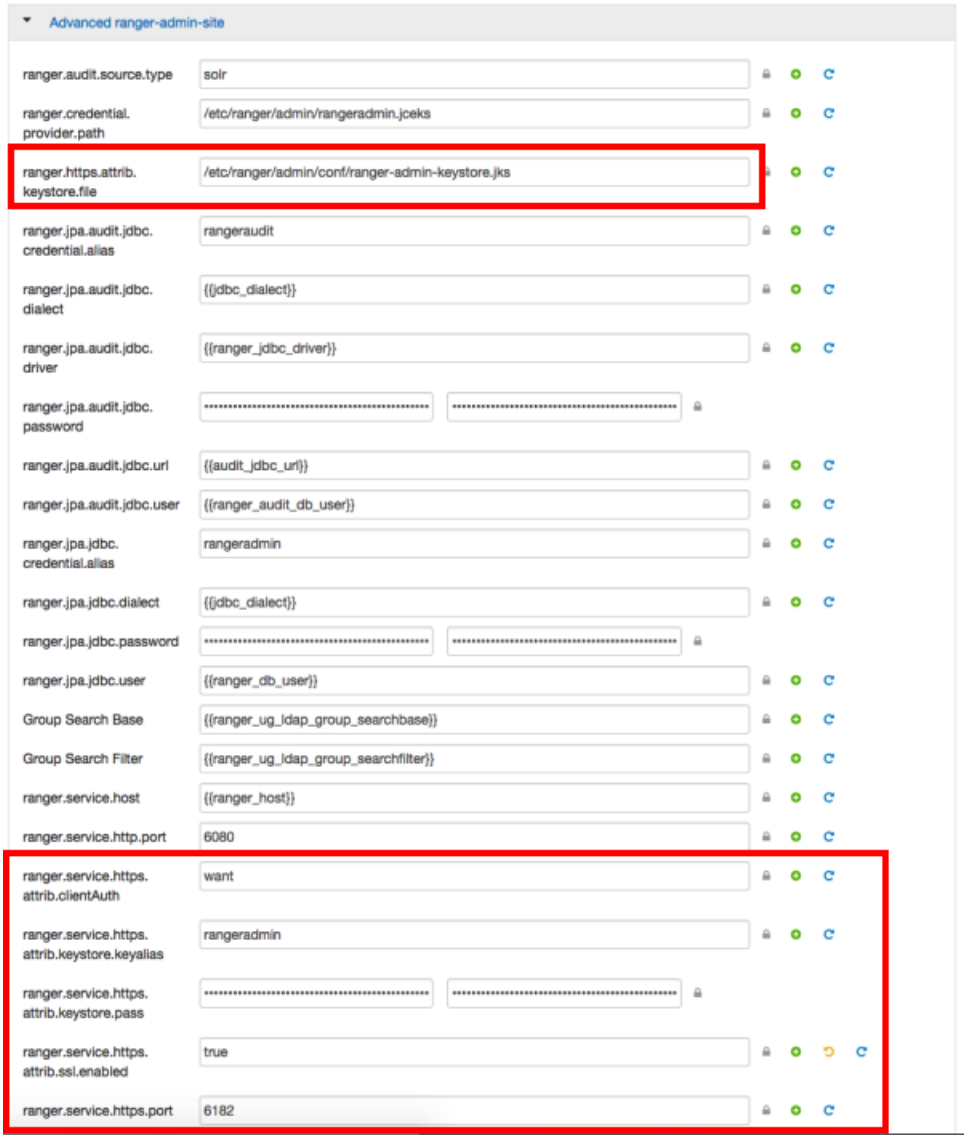


b) Under Ranger Settings, provide the value in the External URL box in the format https://<hostname of policy manager>:<https port>.

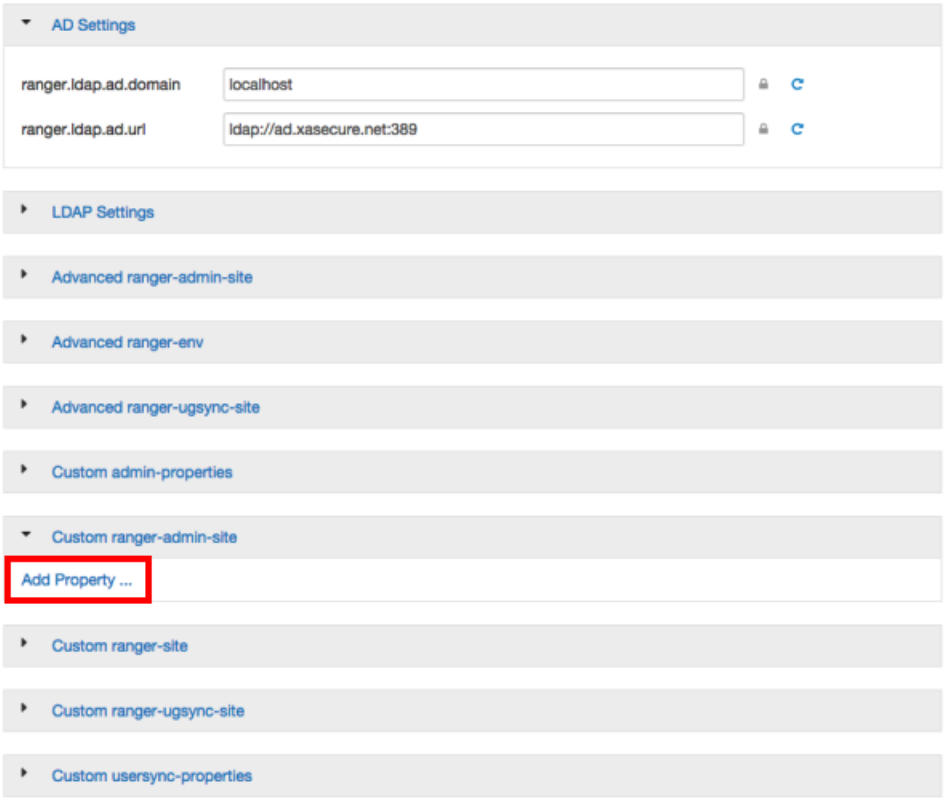


c) Under Advanced ranger-admin-site, set the following properties:

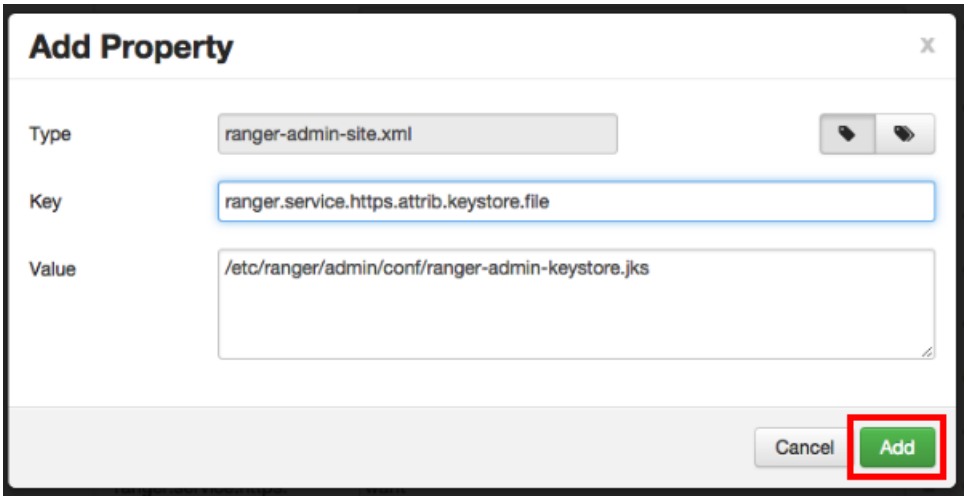
- ranger.https.attrib.keystore.file -- Provide the location of the keystore file created previously: /etc/ranger/admin/conf/ranger-admin-keystore.jks.
- ranger.service.https.attrib.keystore.pass -- Enter the password for the keystore (in this case, xasecure).
- ranger.service.https.attrib.keystore.keyalias -- Enter the alias name for the keystore private key (in this case, rangeradmin).
- ranger.service.https.attrib.clientAuth -- Enter want as the value. This validates the client cert from all agents, but not the requests from web applications. Setting this value to want requires the client to have a certificate to use to sign traffic. If you do not want to put certificates on the client machines to do two-way SSL, this parameter can be set to false to enable one-way SSL.
- ranger.service.https.attrib.ssl.enabled -- set this property to true.
- ranger.service.https.port -- Make sure that this port is available, or change the value to an available port number.



4. Under Custom ranger-admin-site, add the following properties:
   - ranger.service.https.attrib.keystore.file -- Specify the same value provided for the ranger.https.attrib.keystore.file property.
   - ranger.service.https.attrib.client.auth -- Specify the same value provided for the ranger.service.https.attrib.clientAuth property.
5. To add a Custom ranger-admin-site property:
   a) Select Custom ranger-admin-site, then click Add Property.



b) On the Add Property pop-up, type the property name in the Key box, type the property value in the Value box, then click Add.



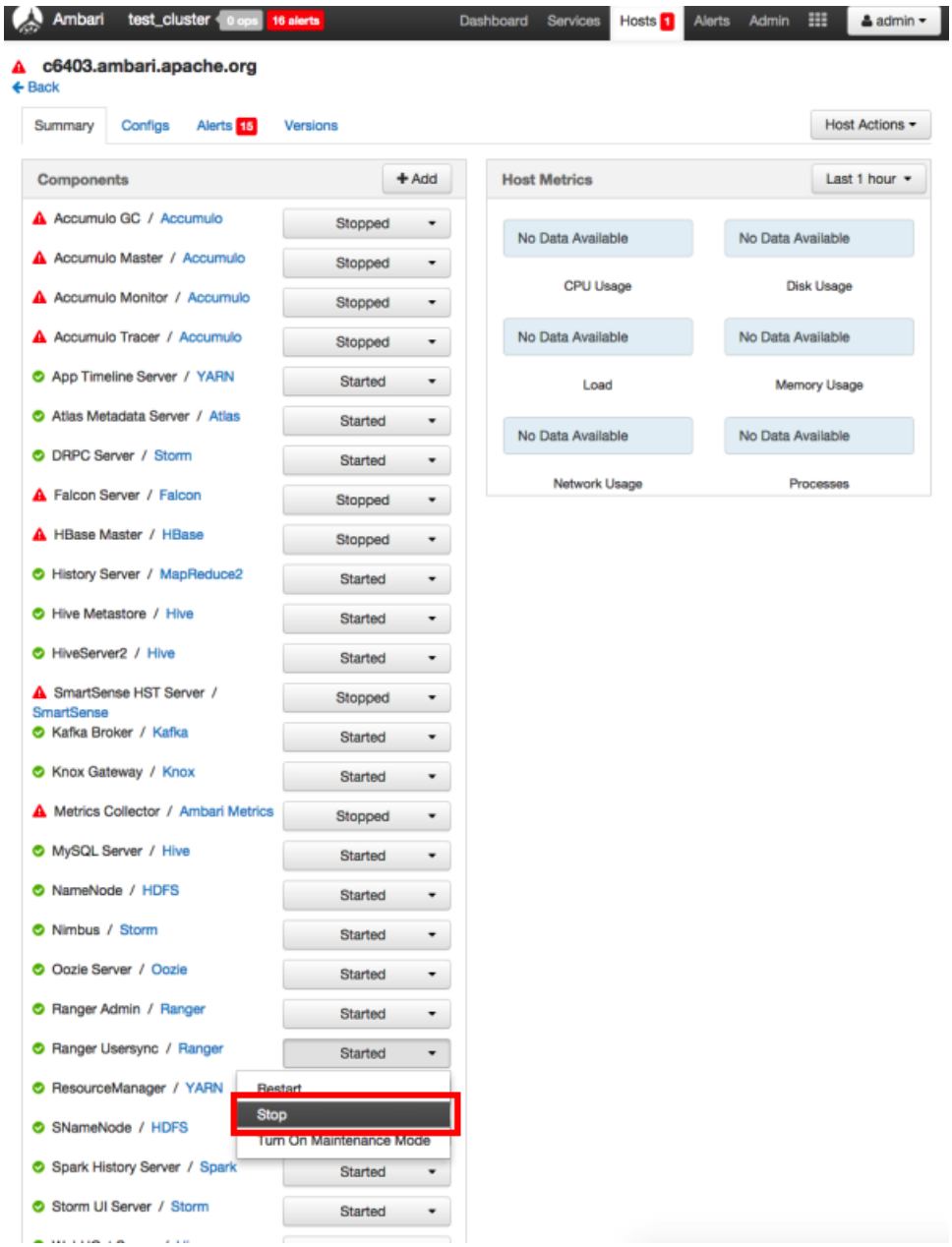
6. Save your changes and start Ranger Admin.
   When you attempt to access the Ranger Admin UI with the HTTPS protocol on the port specified by the ranger.service.https.port property, the browser should report that it does not trust the site. Click Proceed anyway and you should be able to access the Ranger Admin UI over HTTPS.

Configure Ranger Usersync

How to configure Ranger Usersync, when setting up Ambari Ranger SSL using self-signed certificates.

Procedure

1. Stop Ranger Usersync by selecting the Ranger Usersync link, then select Started > Stop next to Ranger Usersync.



2. Check to see if unixauthservice.jks is in the /etc/ranger/usersync/conf/ directory. If not, run the following commands in the CLI:

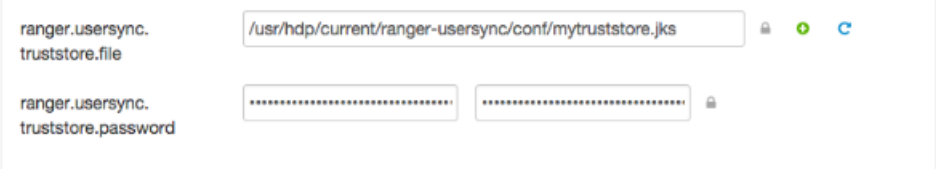
```
cd /etc/ranger/usersync/conf/
mkdir cert
keytool -genkeypair -keyalg RSA -alias selfsigned -keystore /
etc/ranger/usersync/conf/cert/unixauthservice.jks -keypass
 UnIx529p -storepass UnIx529p -validity 3600 -keysize 2048 -dname
 'cn=unixauthservice,ou=authenticator,o=mycompany,c=US'
chown -R ranger:ranger /etc/ranger/usersync/conf/cert
chmod -R 400 /etc/ranger/usersync/conf/cert
```

3. Create a truststore for the Ranger Admin's self-signed keystore. When prompted for a password, press the Enter key.

```
cd /etc/ranger/usersync/conf/
```

```
keytool -export -keystore /etc/ranger/admin/conf/ranger-admin-keystore.jks
 -alias rangeradmin -file ranger-admin-trust.cerchown -R ranger:ranger /
etc/ranger/usersync/conf/cert
keytool -import -file ranger-admin-trust.cer -alias rangeradmintrust -
keystore mytruststore.jks -storepass changeit
chown ranger:ranger mytruststore.jks
```

4. Navigate back to Ranger and select Configs > Advanced, then click Advanced ranger-ugsync-site. Set the following properties:
   - ranger.usersync.truststore.file -- Enter the path to the truststore file.
   - ranger.usersync.truststore.password -- Enter the truststore password.



5. Start Ranger Usersync by selecting the Ranger Usersync link on the Summary tab, then select Stopped > Start next to Ranger Usersync.

Configuring Ranger Plugins (Self-Signed Certificate)

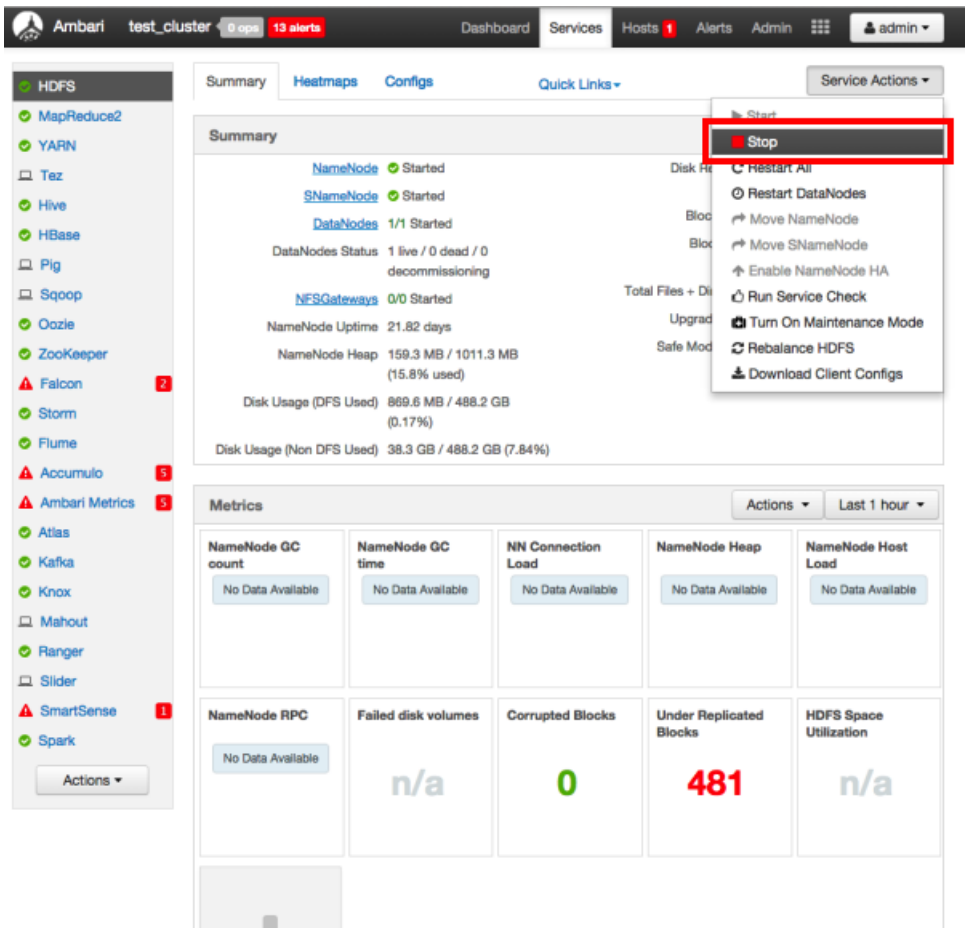
The following section shows how to configure the Ranger HDFS plugin for SSL with a self-signed certificate. You can use the same procedure for other Ranger components. Additional steps required to configure the Ranger KMS plugin and server are provided in subsequent sections.

Configure the Ranger HDFS Plugin for SSL

How to configure the Ranger HDFS Plugin for SSL, when setting up Ambari Ranger SSL using self-signed certificates. The following steps show how to configure the Ranger HDFS plugin for SSL. You can use the same procedure for other Ranger components.

Procedure

1. Stop HDFS by selecting Service Actions > Stop.



2. Change to the Ranger HDFS plugin directory and create a self-signed certificate.

```
cd /etc/hadoop/conf
keytool -genkey -keyalg RSA -alias rangerHdfsAgent -keystore ranger-
plugin-keystore.jks -storepass myKeyFilePassword -validity 360 -keysize
 2048
chown hdfs:hdfs ranger-plugin-keystore.jks
chmod 400 ranger-plugin-keystore.jks
```

3. When prompted, provide an identifiable string as the value for the "What is your first and last name?" question. then provide answers to the subsequent questions to create the keystore. When prompted for a password, press the Enter key.

Note:

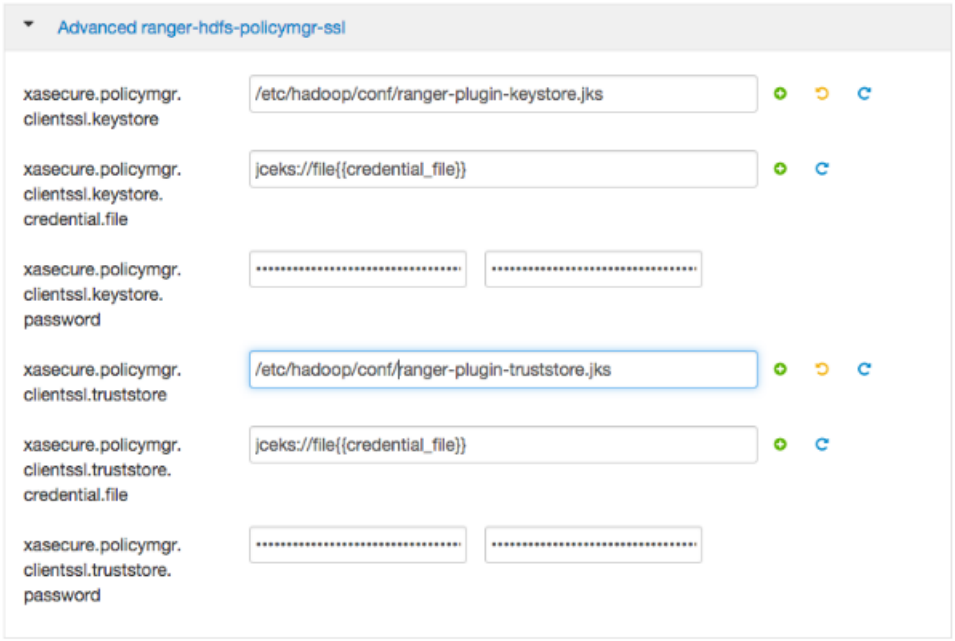
Important note: In the case where multiple servers talking to ranger admin for downloading policies for the same service/repository (e.g. HBase Master and Region servers, multiple NameNodes or Hive servers in an HA environment, be sure to use the repo name or a common string across all of the nodes (such as HbasePlugin, HdfsPlugin etc). (Note this and enter the same value in Common Name For Certificate field in the edit repository page in the Policy Manager UI).

4. Create a truststore for the agent and add the Admin public key as a trusted entry. When prompted for a password, press the Enter key.

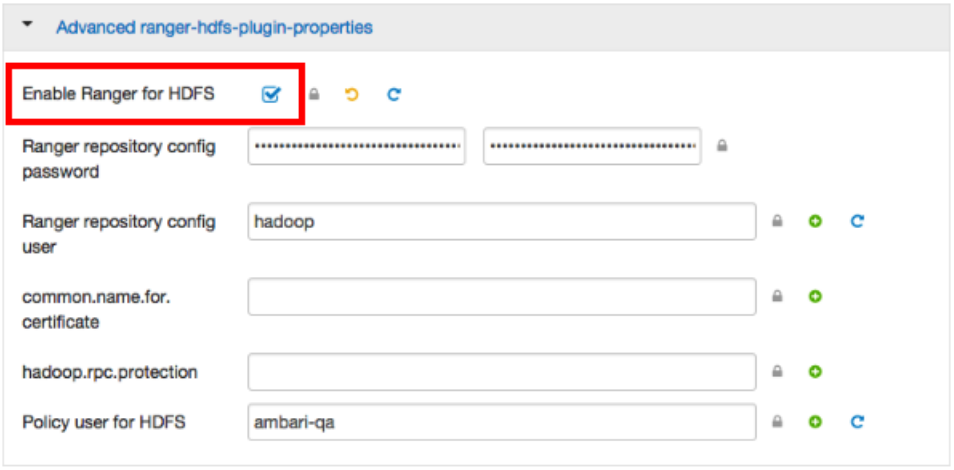
```
cd /etc/hadoop/conf
keytool -export -keystore /etc/ranger/admin/conf/ranger-admin-keystore.jks
 -alias rangeradmin -file ranger-admin-trust.cer
keytool -import -file ranger-admin-trust.cer -alias rangeradmintrust -
keystore ranger-plugin-truststore.jks -storepass changeit
chown hdfs:hdfs ranger-plugin-truststore.jks
```

```
chmod 400 ranger-plugin-truststore.jks
```

5. Under Ranger Settings, provide the value in the External URL box in the format https://<hostname of policy manager>:<https port>.
6. Select Advanced ranger-hdfs-policymgr-ssl and set the following properties:
   - xasecure.policymgr.clientssl.keystore -- Enter the location of the keystore created in the previous step.
   - xasecure.policymgr.clientssl.keystore.password -- Enter the keystore password.
   - xasecure.policymgr.clientssl.truststore -- Enter the location of the truststore created in the previous step.
   - xasecure.policymgr.clientssl.truststore.password -- Enter the truststore password.



7. Select Advanced ranger-hdfs-plugin-properties, then select the Enable Ranger for HDFS check box.



8. Click Save.
9. Start HDFS by selecting Service Actions > Start.
10. Stop Ranger Admin by selecting Service Actions > Stop.
11. Add the agent's self-signed cert to the Admin's trustedCACerts.

```
cd /etc/ranger/admin/conf
```

```
keytool -export -keystore /etc/hadoop/conf/ranger-plugin-keystore.jks
 -alias rangerHdfsAgent -file ranger-hdfsAgent-trust.cer -storepass
 myKeyFilePassword
keytool -import -file ranger-hdfsAgent-trust.cer -alias
 rangerHdfsAgentTrust -keystore <Truststore file used by Ranger Admin -
 can be the JDK cacerts> -storepass changeit
```

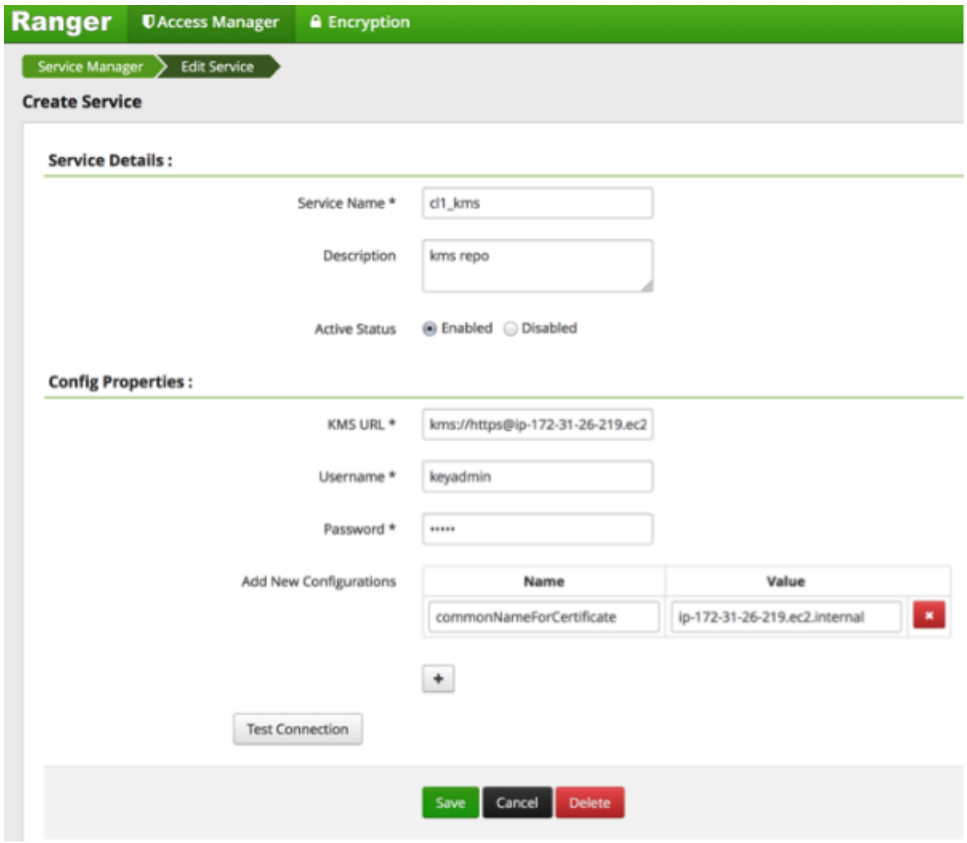
12. Restart Ranger Admin.
13. Log into the Ranger Policy Manager UI as the admin user. Click the Edit button of your repository (in this case, hadoopdev) and provide the CN name of the keystore as the value for Common Name For Certificate, then save your changes.
14. Start the HDFS service.
15. In the Policy Manager UI, select Audit > Plugins.
    You should see an entry for your repo name with HTTP Response Code 200.

Configure the Ranger KMS Plugin for SSL

How to configure the Ranger KMS Plugin for SSL, when setting up Ambari Ranger SSL using self-signed certificates. To configure the Ranger KMS (Key Management Service) plugin for SSL, use the procedure described in the task above, and then perform the following additional step.

Procedure

1. Complete “Configure the Ranger HDFS Plugin for SSL” (link below), modified for HBase.
2. Log into the Policy Manager UI (as the keyadmin user) and click the Edit button of your KMS repository. Provide the CN name of the keystore as the value for Common Name For Certificate and save your changes. This property is not provided by default, so it must be added.



Related Information

Configure the Ranger HDFS Plugin for SSL

Configure the Ranger KMS Server for SSL

How to configure the Ranger KMS Server for SSL, when setting up Ambari Ranger SSL using self-signed certificates.

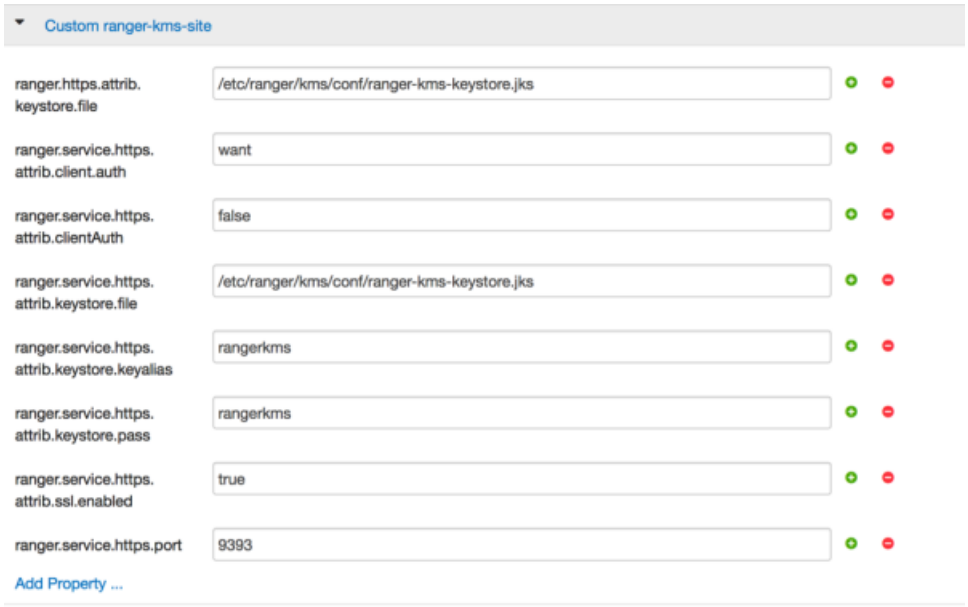
Procedure

1. Stop Ranger KMS by selecting Service Actions > Stop.
2. Change to the Ranger KMS configuration directory and create a self-signed certificate.

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

   a) When prompted, provide an identifiable string as the value for the "What is your first and last name?" question. then provide answers to the subsequent questions to create the keystore.
   b) When prompted for a password, press the Enter key.
3. Select Custom ranger-kms-site, then add the following properties as shown below:

   - ranger.https.attrib.keystore.file
   - ranger.service.https.attrib.keystore.file (duplicate of above – workaround for now)
   - ranger.service.https.attrib.clientAuth
   - ranger.service.https.attrib.client.auth (duplicate of above – workaround for now)
   - ranger.service.https.attrib.keystore.keyalias
   - ranger.service.https.attrib.keystore.pass
   - ranger.service.https.attrib.ssl.enabled
   - ranger.service.https.port



4. Under Advanced kms_env, update the value of kms_port to match the value of ranger.service.https.port.
5. Save your changes and restart Ranger KMS.
   When you attempt to access the Ranger KMS UI with the HTTPS protocol on the port specified by the ranger.service.https.port property, the browser should report that it does not trust the site. Click Proceed anyway and you should be able to access the Ranger Admin UI over HTTPS.

6. Export the Ranger KMS certificate.

```
cd /usr/hdp/<version>/ranger-kms/conf
keytool -export -keystore ranger-kms-keystore.jks -alias rangerkms -file
 ranger-kms-trust.cer
```

7. Import the Ranger KMS certificate into the Ranger Admin truststore.

```
keytool -import -file ranger-kms-trust.cer -alias rangerkms -keystore
 <Truststore file used by Ranger Admin - can be the JDK cacerts> -
storepass changeit
```

8. Import the Ranger KMS certificate into the Hadoop client truststore.

```
keytool -import -file ranger-kms-trust.cer -alias rangerkms -keystore /
etc/security/clientKeys/all.jks -storepass bigdata
```

9. Restart Ranger Admin and HDFS.

Configure Ranger Admin Database for SSL-Enabled MySQL (Ranger SSL)

When an SSL-enabled database is configured for use with Ranger, you must add certain configurations to Ranger

Procedure

1. In Ambari>Ranger>Configs>Advanced>Custom admin-properties, add the following parameters:
   - db_ssl_enabled=True
   - db_ssl_required=True
   - db_ssl_verifyServerCertificate=True
   - javax_net_ssl_keyStore=/etc/ranger/admin/keystore
   - javax_net_ssl_keyStorePassword=ranger
   - javax_net_ssl_trustStore=/etc/ranger/admin/truststore
   - javax_net_ssl_trustStorePassword=ranger

   Change keystore and truststore file paths according to your environment.

   If certificate verification is not required, you can set value false in property db_ssl_verifyServerCertificate. In this case, keystore and truststore file location need not to be valid and/or mandatory.
2. In Ambari>Ranger>Configs>Advanced>Custom ranger-admin-site, add the following parameters:
   - ranger.db.ssl.enabled=true
   - ranger.db.ssl.required=true
   - ranger.db.ssl.verifyServerCertificate=true
   - ranger.keystore.file=/etc/ranger/admin/keystore
   - ranger.keystore.password=ranger

   Change keystore file path according to your environment.

   If certificate verification is not required, then you can set value false in property ranger.db.ssl.verifyServerCertificate. In this case, keystore and truststore file location need not to be valid and/or mandatory.
3. In Ambari>Ranger>Configs>Advanced>Advanced ranger-admin-site, add the following parameters:
   - ranger.truststore.file=/etc/ranger/admin/truststore
   - ranger.truststore.password=password
4. Install/restart Ranger.

Connecting to SSL Enabled Components

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

Connect to SSL-Enabled HiveServer2 using JDBC

How to connect to SSL-enabled HiveServer2 using JDBC.

About this task

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

Tip:

See HIVE-4911 for more details on this enhancement.

Procedure

From the JDBC client specify sasl.sop as part of the JDBC-Hive connection string.
jdbc:hive://hostname/dbname;sasl.qop=auth-int

Related Information

Locate the JDBC or ODBC driver
HIVE-4911

Connecting to SSL Enabled Oozie Server

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

Use a Self-Signed Certificate from Oozie Java Clients

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

Procedure

1. Install the certificate in the keychain:
   a) Copy or download the .cert file onto the client machine.
   b) Run the following command (as root) to import the certificate into the JRE's keystore: sudo keytool -import -alias tomcat -file path/to/certificate.cert -keystore <JRE_cacerts>.

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

      Java programs, including the Oozie client, can now connect to the Oozie Server using the self-signed certificate.
2. In the connection strings change HTTP to HTTPS, for example, replace http://oozie.server.hostname:11000/oozie with https://oozie.server.hostname:11443/oozie.

   Java does not automatically redirect HTTP addresses to HTTPS.

Connect to Oozie from Java Clients

How to connect to Oozie from Java clients.

About this task

Java does not automatically redirect HTTP addresses to HTTPS.

Procedure

In the connection strings change HTTP to HTTPS and adjust the port.
Replace http://oozie.server.hostname:11000/oozie with https://oozie.server.hostname:11443/oozie.

Connect to Oozie from a Web Browser

How to connect to Oozie from a web browser.

Procedure

Use https://oozie.server.hostname:11443/oozie though most browsers should automatically redirect you if you use http://oozie.server.hostname:11000/oozie.
When using a Self-Signed Certificate, your browser warns you that it can't verify the certificate. Add the certificate as an exception.