Configuring Authentication with Kerberos

Date of Publish: 2018-07-15

http://docs.hortonworks.com

Contents

Configuring Authentication with Kerberos..... 3

Kerberos Overview..... 3

Kerberos Principals Overview..... 4

Enabling SPNEGO Authentication for Hadoop..... 5

Set Up Kerberos for Ambari Server..... 5
Configure HTTP Authentication for HDFS, YARN, MapReduce2, HBase, Oozie, Falcon, and Storm..... 6
Enable Browser Access to a SPNEGO-enabled Web UI..... 7

Enabling Kerberos Authentication Using Ambari..... 7

Checklist: Installing and Configuring the KDC..... 7
Optional: Install a new MIT KDC..... 8
Optional: Use an Existing IPA..... 10
Install the JCE for Kerberos..... 11
Enabling Kerberos Security..... 12
Create Mappings Between Principals and UNIX Usernames..... 13
Running the Kerberos Security Wizard..... 14
Update KDC Admin Credentials..... 17
Customizing the Attribute Template..... 18
Disable Kerberos Security..... 18

Configuring HDP Components for Kerberos..... 19

Configuring Kafka for Kerberos..... 19
Kerberos for Kafka Prerequisites..... 19
Configuring the Kafka Broker for Kerberos..... 19
Create Kafka Topics..... 20
Produce Events or Messages to Kafka on a Secured Cluster..... 20
Consume Events or Messages from Kafka on a Secured Cluster..... 22
Authorizing Access when Kerberos is Enabled..... 25
Appendix: Kerberos Kafka Configuration Options..... 25
Configuring Storm for Kerberos..... 28
Kerberos for Storm Prerequisites..... 28
Designating a Storm Client Node..... 28
Running Workers as Users..... 30
Accessing the Storm UI..... 30
Accessing the Storm UI Active Directory Trust Configuration..... 30
Kerberos Storm Security Properties..... 31
Known Issues with Storm for Kerberos..... 32
Securing Apache HBase in a production environment..... 33
Installing Apache HBase with Kerberos on an existing HDP cluster..... 33
Verify if kerberos is enabled for HBase..... 34
Access Kerberos-enabled HBase cluster using a Java client..... 37

Configuring Authentication with Kerberos

Establishing user identity with strong authentication is the basis for secure access in Hadoop. Users need to reliably identify themselves and then have that identity propagated throughout the Hadoop cluster to access cluster resources. Hortonworks uses Kerberos for authentication.

Kerberos is an industry standard used to authenticate users and resources within a Hadoop cluster. HDP also includes Ambari, which simplifies Kerberos setup, configuration, and maintenance.

By default Ambari requires that a user authenticate using a user name and password. Ambari uses this authentication mechanism whether you configure it to authenticate using its internal database or synchronized with an external source, like LDAP or Active Directory. Optionally, you can configure Ambari to authenticate using Kerberos tokens via SPNEGO (Simple and Protected GSSAPI Negotiation Mechanism).

Kerberos Overview

A high-level overview of Kerberos authentication, including the Key Distribution Center, principals, Authentication Server, Ticket Granting Tickets, and Ticket Granting Server.

Strongly authenticating and establishing a user’s identity is the basis for secure access in Hadoop. Users need to be able to reliably “identify” themselves and then have that identity propagated throughout the Hadoop cluster. Once this is done, those users can access resources (such as files or directories) or interact with the cluster (like running MapReduce jobs). Besides users, Hadoop cluster resources themselves (such as Hosts and Services) need to authenticate with each other to avoid potential malicious systems or daemon’s “posing as” trusted components of the cluster to gain access to data.

Hadoop uses Kerberos as the basis for strong authentication and identity propagation for both user and services. Kerberos is a third party authentication mechanism, in which users and services rely on a third party - the Kerberos server - to authenticate each to the other. The Kerberos server itself is known as the Key Distribution Center, or KDC. At a high level, it has three parts:

- A database of the users and services (known as principals) that it knows about and their respective Kerberos passwords
- An Authentication Server (AS) which performs the initial authentication and issues a Ticket Granting Ticket (TGT)
- A Ticket Granting Server (TGS) that issues subsequent service tickets based on the initial TGT

A user principal requests authentication from the AS. The AS returns a TGT that is encrypted using the user principal's Kerberos password, which is known only to the user principal and the AS. The user principal decrypts the TGT locally using its Kerberos password, and from that point forward, until the ticket expires, the user principal can use the TGT to get service tickets from the TGS. Service tickets are what allow a principal to access various services.

Because cluster resources (hosts or services) cannot provide a password each time to decrypt the TGT, they use a special file, called a keytab , which contains the resource principal's authentication credentials. The set of hosts, users, and services over which the Kerberos server has control is called a realm.

Terminology

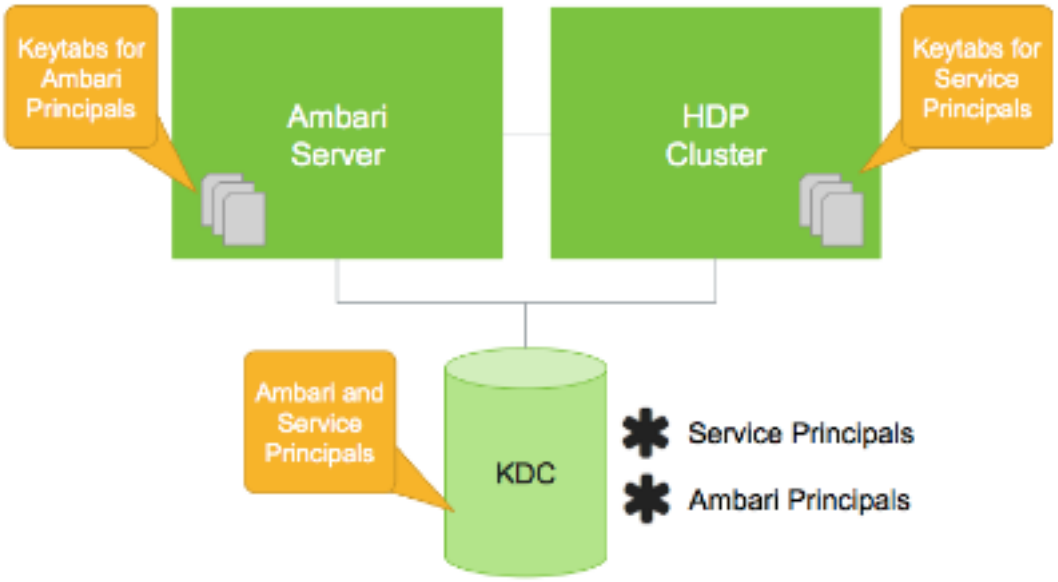
Term	Description
Key Distribution Center, or KDC	The trusted source for authentication in a Kerberos-enabled environment.
Kerberos KDC Server	The machine, or server, that serves as the Key Distribution Center (KDC).
Kerberos Client	Any machine in the cluster that authenticates against the KDC.
Principal	The unique name of a user or service that authenticates against the KDC.

Term	Description
Keytab	A file that includes one or more principals and their keys.
Realm	The Kerberos network that includes a KDC and a number of Clients.
KDC Admin Account	An administrative account used by Ambari to create principals and generate keytabs in the KDC.

Kerberos Principals Overview

A conceptual overview of Kerberos principals.

Each service and sub-service in Hadoop must have its own principal. A principal name in a given realm consists of a primary name and an instance name, in this case the instance name is the FQDN of the host that runs that service. As services do not log in with a password to acquire their tickets, their principal's authentication credentials are stored in a keytab file, which is extracted from the Kerberos database and stored locally in a secured directory with the service principal on the service component host.



Principal and Keytab Naming Conventions

Asset	Convention	Example
Principals	$service_component_name/$FQDN@EXAMPLE.COM	nn/c6401.ambari.apache.org@EXAMPLE.COM
Keytabs	$service_component_abbreviation.service.keytab	/etc/security/keytabs/nn.service.keytab

Notice in the preceding example the primary name for each service principal. These primary names, such as nn or hive for example, represent the NameNode or Hive service, respectively. Each primary name has appended to it the instance name, the FQDN of the host on which it runs. This convention provides a unique principal name for services that run on multiple hosts, like DataNodes and NodeManagers. Adding the host name serves to distinguish, for example, a request from DataNode A from a request from DataNode B. This is important for the following reasons:

- Compromised Kerberos credentials for one DataNode do not automatically lead to compromised Kerberos credentials for all DataNodes.

- If multiple DataNodes have exactly the same principal and are simultaneously connecting to the NameNode, and if the Kerberos authenticator being sent happens to have same timestamps, then the authentication is rejected as a replay request.

In addition to the Hadoop Service Principals, Ambari itself also requires a set of Ambari Principals to perform service "smoke" checks, perform alert health checks and to retrieve metrics from cluster components. Keytab files for the Ambari Principals reside on each cluster host, just as keytab files for the service principals.

Ambari Principals	Description
Smoke and "Headless" Service users	Used by Ambari to perform service "smoke" checks and run alert health checks.
Ambari Server user	When a cluster is enabled for Kerberos, the component REST endpoints (such as the YARN ATS com SPNEGO authentication. Ambari Server needs access to these APIs and requires a Kerberos principal authenticate via SPNEGO against these APIs.

Related Information

Enabling SPNEGO Authentication for Hadoop

Enabling SPNEGO Authentication for Hadoop

By default, access to the HTTP-based services and UIs for the cluster are not configured to require authentication. Kerberos authentication can be configured for the Web UIs for HDFS, YARN, MapReduce2, HBase, Oozie, Falcon and Storm.

Set Up Kerberos for Ambari Server

This section describes how to configure Ambari Server with a Kerberos principal and keytab to allow views to authenticate via SPNEGO against cluster components.

About this task

When a cluster is enabled for Kerberos, the component REST endpoints (such as the YARN ATS component) require SPNEGO authentication.

Depending on the Services in your cluster, Ambari Web needs access to these APIs. As well, some views need access to ATS. Therefore, the Ambari Server requires a Kerberos principal in order to authenticate via SPNEGO against these APIs. This section describes how to configure Ambari Server with a Kerberos principal and keytab to allow views to authenticate via SPNEGO against cluster components.

Procedure

1. Create a principal in your KDC for the Ambari Server. For example, using kadmin. addprinc -randkey ambari-server@EXAMPLE.COM.
2. Generate a keytab for that principal. xst -k ambari.server.keytab ambari-server@EXAMPLE.COM.
3. Place that keytab on the Ambari Server host. Be sure to set the file permissions so the user running the Ambari Server daemon can access the keytab file. /etc/security/keytabs/ambari.server.keytab.
4. Stop the ambari server. ambari-server stop.
5. Run the setup-security command. ambari-server setup-security.
6. Select 3 for Setup Ambari kerberos JAAS configuration.
7. Enter the Kerberos principal name for the Ambari Server you set up earlier.
8. Enter the path to the keytab for the Ambari principal.
9. Restart Ambari Server. ambari-server restart.

Configure HTTP Authentication for HDFS, YARN, MapReduce2, HBase, Oozie, Falcon, and Storm

How to configure HTTP authentication for Hadoop components in a Kerberos environment.

Procedure

1. Create a secret key used for signing authentication tokens. This file should contain random data and be placed on every host in the cluster. It should also be owned by the hdfs user and group owned by the hadoop group. Permissions should be set to 440. For example:

```
dd if=/dev/urandom of=/etc/security/http_secret bs=1024 count=1
chown hdfs:hadoop /etc/security/http_secret
chmod 440 /etc/security/http_secret
```

2. In Ambari Web, browse to Services > HDFS > Configs .
3. Add or modify the following configuration properties to Advanced core-site:

Property	New Value
hadoop.http.authentication.simple.anonymous.allowed	false
hadoop.http.authentication.signature.secret.file	/etc/security/http_secret
hadoop.http.authentication.type	kerberos
hadoop.http.authentication.kerberos.keytab	/etc/security/keytabs/spnego.service.keytab
hadoop.http.authentication.kerberos.principal	HTTP/_HOST@ EXAMPLE.COM
hadoop.http.filter.initializers	org.apache.hadoop.security.AuthenticationFilterInitializer
hadoop.http.authentication.cookie.domain	hortonworks.local

Note:

The entries listed in the above table in bold and italicized are site-specific. The hadoop.http.authentication.cookie.domain property is based off of the fully qualified domain names of the servers in the cluster. For example if the FQDN of your NameNode is host1.hortonworks.local, the hadoop.http.authentication.cookie.domain should be set to hortonworks.local.

4. For HBase, you can enable Kerberos-authentication to HBase Web UIs by configuring SPNEGO.
   a) In Ambari Web, browse to Services > HBase > Configs .
   b) Add the following configuration properties to the custom hbase-site.xml file:

Property	Value
hbase.security.authentication.ui	kerberos
hbase.security.authentication	kerberos
hbase.security.authentication.spnego.kerberos.principal	HTTP/_HOST@EXAMPLE.COM
hbase.security.authentication.spnego.kerberos.keytab	/etc/security/keytabs/spnego.service.keytab
Hbase.security.authentication.spnego.kerberos.name.rules (Optional)	
Hbase.security.authentication.signature.secret.file(Optional)	

5. Save the configuration, then restart the affected services.

Enable Browser Access to a SPNEGO-enabled Web UI

How to enable browser access to a SPNEGO-enabled web UI.

Procedure

1. Install Kerberos on your local machine (search for instructions on how to install a Kerberos client on your local environment).
2. Configure the krb5.conf file on your local machine. For testing on a HDP cluster, copy the /etc/krb5.conf file from one of the cluster hosts to your local machine at /etc/krb5.conf. Create your own keytabs and run kinit. For testing on a HDP cluster, copy the "ambari_qa" keytab file from /etc/security/keytabs/smokeuser.headless.keytab on one of the cluster hosts to your local machine, then run the following command:

   ```
   kinit -kt smokeuser.headless.keytab ambari-qa@EXAMPLE.COM
   ```

3. Enable your web browser with Kerberos SPNEGO:
   a) For Chrome on Mac:
      1. Run the following command from the same shell in which you ran the previous kinit command to launch Chrome:

         ```
         /Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --
         auth-server-whitelist="*.hwx.site"
         ```

         Replace .hwx.site with your own domain name.
      2. If you get the following error, try closing and relaunching all Chrome browser windows.

         ```
         [14617:36099:0810/152439.802775:ERROR:browser_gpu_channel_host_factory.cc(103)]
          Failed to launch GPU process.
         ```

   b) For Firefox:
      1. Navigate to the about:config URL (type about:config in the address box, then press the Enter key).
      2. Scroll down to network.negotiate-auth.trusted-uris and change its value to your cluster domain name (For example, .hwx.site).
      3. Change the value of network.negotiate-auth.delegation-uris to your cluster domain name (For example, .hwx.site).

Enabling Kerberos Authentication Using Ambari

This section describes how to configure Kerberos for strong authentication for Hadoop users and hosts in an Ambari-managed cluster.

Checklist: Installing and Configuring the KDC

Ambari is able to configure Kerberos in the cluster to work with an existing MIT KDC, or existing Active Directory installation. This section describes the steps necessary to prepare for this integration.

You can choose to have Ambari connect to the KDC and automatically create the necessary Service and Ambari principals, generate and distribute the keytabs (“Automated Kerberos Setup”). Ambari also provides an advanced option to manually configure Kerberos. If you choose this option, you must create the principals, generate and distribute the keytabs. Ambari will not do this automatically (“Manual Kerberos Setup”).

Supported Key Distribution Center (KDC) Versions

- Microsoft Active Directory 2008 and above

- MIT Kerberos v5
- FreeIPA 4.x and above

There are four ways to install/configure the KDC:

- Using an existing MIT KDC
- Install a new MIT KDC (See "Optional: Install a new MIT KDC")
- Using an existing IPA
- Using an existing AD
- Using manual Kerberos setup

Option	Checklist
Using an existing MIT KDC	• Ambari Server and cluster hosts have network access to both the KDC and KDC admin hosts. • KDC administrative credentials are on-hand.
Install a new MIT KDC	See “Optional: Install a new MIT KDC”
Using an existing IPA	See “Optional: Use an Existing IPA”
Using an existing AD	• Ambari Server and cluster hosts have network access to, and be able to resolve the DNS names of, the Domain Controllers. • Active Directory secure LDAP (LDAPS) connectivity has been configured. • Active Directory User container for service principals has been created and is on-hand. For example, "OU=Hadoop,OU=People,dc=apache,dc=org" • Active Directory administrative credentials with delegated control of “Create, delete, and manage user accounts” on the previously mentioned User container are on-hand.
Using manual Kerberos setup	• Cluster hosts have network access to the KDC. • Kerberos client utilities (such as kinit) have been installed on every cluster host. • The Java Cryptography Extensions (JCE) have been setup on the Ambari Server host and all hosts in the cluster. • The Service and Ambari Principals will be manually created in the KDC before completing this wizard. • The keytabs for the Service and Ambari Principals will be manually created and distributed to cluster hosts before completing this wizard.

Related Information

Enabling Kerberos Security
Optional: Install a new MIT KDC

Optional: Install a new MIT KDC

The following gives a very high level description of the KDC installation process.

About this task

To get more information see specific Operating Systems documentation, such as RHEL documentation, CentOS documentation, or SLES documentation (links below).

Procedure

1. Install the KDC Server:
   a) Install a new version of the KDC server:

OS Flavor	Enter
RHEL/CentOS/Oracle Linux	yum install krb5-server krb5-libs krb5-workstation
SLES	zypper install krb5 krb5-server krb5-client
Ubuntu/Debian	apt-get install krb5-kdc krb5-admin-server

b) Using a text editor, open the KDC server configuration file, located by default here: vi /etc/krb5.conf.

c) Change the [realms] section of this file by replacing the default “kerberos.example.com” setting for the kdc and admin_server properties with the Fully Qualified Domain Name of the KDC server host. In the following example, “kerberos.example.com” has been replaced with “my.kdc.server”.

```
realms]
 EXAMPLE.COM = {
   kdc = my.kdc.server
   admin_server = my.kdc.server
}
```

2. Use the utility kdb5_util to create the Kerberos database:

OS Flavor	Enter
RHEL/CentOS/Oracle Linux	kdb5_util create -s
SLES	kdb5_util create -s
Ubuntu/Debian	krb5_newrealm

3. Start the KDC server and the KDC admin server:

OS Flavor	Enter
RHEL/CentOS/Oracle Linux 6	/etc/rc.d/init.d/krb5kdc start /etc/rc.d/init.d/kadmin start
RHEL/CentOS/Oracle Linux 7	systemctl start krb5kdc systemctl start kadmin
SLES	rckrb5kdc start rckadmind start
Ubuntu/Debian	service krb5-kdc restart service krb5-admin-server restart

4. Set up the KDC server to auto-start on boot:

OS Flavor	Enter
RHEL/CentOS/Oracle Linux 6	chkconfig krb5kdc on chkconfig kadmin on
RHEL/CentOS/Oracle Linux 7	systemctl enable krb5kdc systemctl enable kadmin
SLES	chkconfig rckrb5kdc on chkconfig rckadmind on
Ubuntu/Debian	update-rc.d krb5-kdc defaults update-rc.d krb5-admin-server defaults

5. Create a Kerberos Admin:

Kerberos principals can be created either on the KDC machine itself or through the network, using an “admin” principal. The following instructions assume you are using the KDC machine and using the kadmin.local

command line administration utility. Using kadmin.local on the KDC machine allows you to create principals without needing to create a separate "admin" principal before you start.

a) Create a KDC admin by creating an admin principal: kadmin.local -q "addprinc admin/admin".

b) Confirm that this admin principal has permissions in the KDC ACL. Using a text editor, open the KDC ACL file:

OS Flavor	Enter
RHEL/CentOS/Oracle Linux	vi /var/kerberos/krb5kdc/kadm5.acl
SLES	vi /var/lib/kerberos/krb5kdc/kadm5.acl
Ubuntu/Debian	vi /etc/krb5kdc/kadm5.acl

c) Ensure that the KDC ACL file includes an entry so to allow the admin principal to administer the KDC for your specific realm. When using a realm that is different than EXAMPLE.COM, be sure there is an entry for the realm you are using. If not present, principal creation will fail. For example, for an admin/admin@HADOOP.COM principal, you should have an entry: */admin@HADOOP.COM *.

d) After editing and saving the kadm5.acl file, you must restart the kadmin process:

OS Flavor	Enter
RHEL/CentOS/Oracle Linux 6	/etc/rc.d/init.d/kadmin restart
RHEL/CentOS/Oracle Linux 7	systemctl restart kadmin
SLES	rckadmind restart
Ubuntu/Debian	service krb5-admin-server restart

Related Information

RHEL Documentation> Configuring A Kerberos 5 Server

SLES Documentation> Installing and Administering Kerberos

Optional: Use an Existing IPA

You can use an existing FreeIPA setup with Kerberos.

To use an existing IPA KDC with Automated Kerberos Setup, you must prepare the following:

- All cluster hosts should be joined to the IPA domain and registered in DNS- If IPA is not configured to authoritatively manage DNS, explicitly configuring the private IP and corresponding fully qualified domain names of all hosts, in the /etc/hosts file on all the hosts is recommended.
- If you do not plan on using Ambari to manage the krb5.conf file, ensure the following is set in each krb5.conf file in your cluster: default_ccache_name = /tmp/krb5cc_%{uid} - Redhat/Centos 7.x changed the default ticket cache to keyring, which is problematic for the hadoop components.
- The Java Cryptography Extensions (JCE) have been setup on the Ambari Server host and all hosts in the cluster- If during installation you chose to use the Ambari provided JDK, this has already been done for you. If you configured a custom JDK, ensure the unlimited strength JCE policies are in place on all nodes. For more information, refer to “Install the JCE for Kerberos”.

Please also note:

- If you plan on leveraging this IPA to create trusts with other KDCs, please follow the FreeIPA “Considerations for Active Directory integration” to ensure your hosts use a non-overlapping DNS domain, with matching uppercase REALM.
- Kerberos authentication allows maximum 3 seconds time discrepancy. Use of IPA’s NTP server or an external time management service is highly recommended for all cluster hosts, including the FreeIPA host.
- To avoid exposing the IPA admin account, consider creating a dedicated hadoopadmin account that is a member of the admins group, or has been added to a role with User & Service Administratration privileges. Remember to reset the initial temporary password for the account before use in Ambari. For more details on this process see the section below.

Creating an IPA account for use with Ambari

Example creating hadoopadmin account with explicit privileges

```
# obtain valid ticket as IPA administrator
kinit admin

# create a new principal to be used for ambari kerberos administration
ipa user-add hadoopadmin --first=Hadoop --last=Admin --password

# create a role and and give it privilege to manage users and services
ipa role-add hadoopadminrole
ipa role-add-privilege hadoopadminrole --privileges="User Administrators"
ipa role-add-privilege hadoopadminrole --privileges="Service Administrators"

# add the hadoopadmin user to the role
ipa role-add-member hadoopadminrole --users=hadoopadmin

# login once, or kinit, to reset the initial temporary password for the
 hadoopadmin account
kinit hadoopadmin
```

Important: Do not install an Ambari Agent on the IPA host.

- IPA leverages the SPNEGO principal (HTTP/ipa.your.domain.com) for secure access to its Web UI component. Installing the Ambari Agent on the IPA host causes the kvno of SPNEGO principal to increase, which causes problems for IPA HTTP server. If you have already accidentally done this and IPA is not able to start, symlink IPA's http keytab path (/var/lib/ipa/gssproxy/http.keytab) to /etc/security/keytabs/spnego.service.keytab and contact your IPA provider's support.
- The /etc/krb5.conf file on the IPA host has some additional properties not captured in Ambari's krb5.conf template. Since letting Ambari manage krb5.conf on the cluster hosts is recommended, making the IPA host a part of the cluster is problematic for the IPA services. If you had this option checked when the ambari agent was installed, and do not have a backup of the original krb5.conf, reference the "krb5.conf template" to restore immediate functionality.

Related Information

FreeIPA: Deployment recommendations>Considerations for Active Directory Integration
Install the JCE for Kerberos
FreeIPA: IPA Deployment Recommendations
RedHat: IPA Deployment Recommendations
FreeIPA: IPA Installation Instructions
RedHat: IPA Installation Instructions
HCC: Disabling Keyring in Redhat 7.x
Stackoverflow: Disabling Keyring in Redhat 7.x
Setting up trusts between IPA and Active Directory
krb5.conf template

Install the JCE for Kerberos

Before enabling Kerberos in the cluster, you must deploy the Java Cryptography Extension (JCE) security policy files on the Ambari Server and on all hosts in the cluster, including the Ambari Server. If you are using OpenJDK, some distributions of the OpenJDK (such as RHEL/CentOS and Ubuntu) come with unlimited strength JCE automatically and therefore, installation of JCE is not required.

Procedure

1. On the Ambari Server, obtain the JCE policy file appropriate for the JDK version in your cluster:

Option	
Oracle JDK 1.8	JCE Unlimited Strength Jurisdiction Policy Files 8 Download
Oracle JDK 1.7	JCE Unlimited Strength Jurisdiction Policy Files 7 Download

wget --no-check-certificate --no-cookies --header "Cookie: oraclelicense=accept-securebackup-cookie" "http://download.oracle.com/otn-pub/java/jce/8/jce_policy-8.zip"

2. Save the policy file archive in a temporary location.
3. On Ambari Server and on each host in the cluster, add the unlimited security policy JCE jars to $JAVA_HOME/jre/lib/security/.

   For example, run the following to extract the policy jars into the JDK installed on your host:

   ```
   unzip -o -j -q jce_policy-8.zip -d /usr/jdk64/jdk1.8.0_40/jre/lib/security/
   ```

4. Restart Ambari Server: sudo ambari-server restart.

What to do next

Proceed to “Running the Kerberos Security Wizard”.

Related Information

Running the Kerberos Security Wizard

Enabling Kerberos Security

Whether you choose automated or manual Kerberos setup, Ambari provides a wizard to help with enabling Kerberos in the cluster. This section provides information on preparing Ambari before running the wizard, and the steps to run the wizard.

Prerequisites

- Having the JCE installed on all hosts on the cluster (including the Ambari Server).
- Having the Ambari Server host as part of the cluster.
- Create mappings between principals and UNIX user names. . Creating mappings can help resolve access issues related to case mismatches between principal and local user names.

Exclusions

Ambari Metrics will not be secured with Kerberos unless it is configured for distributed metrics storage. By default, it uses embedded metrics storage and will not be secured as part of the Kerberos Wizard. If you wish to have Ambari Metrics secured with Kerberos, please see “Customizing the Metrics Collector Mode” to enable distributed metrics storage prior to running the Kerberos Wizard.

Centrify Server Suite

If Centrify is installed and being used on any of the servers in the cluster, it is critical that you refer to Centrify's integration guide before attempting to enable Kerberos Security on your cluster. The documentation can be found in the Centrify Server Suite documentation library. A direct link to the Hortonworks-specific configuration guide can be found below.

Related Information

Create Mappings Between Principals and UNIX Usernames

Centrify Server Suite 2016: Centrify Identity and Access Management for Hortonworks

Tuning Performance for AMS>Customize AMS collector mode
Disable Kerberos Security

Create Mappings Between Principals and UNIX Usernames

HDP uses a rule-based system to create mappings between service principals and their related UNIX usernames. The rules are specified in the core-site.xml configuration file as the value to the optional key hadoop.security.auth_to_local.

About this task

The default rule is simply named DEFAULT. It translates all principals in your default domain to their first component. For example, myusername@APACHE.ORG and myusername/admin@APACHE.ORG both become myusername, assuming your default domain is APACHE.ORG.

While mapping the Kerberos principals, if the Kerberos principal names are in the UPPERCASE or CaMeLcase, the names will not be recognized on the Linux machine (as Linux users are always in lower case). You must add the extra switch "/L" in the rule definition to force the conversion to lower case.

Creating Rules

To accommodate more complex translations, you can create a hierarchical set of rules to add to the default. Each rule is divided into three parts: base, filter, and substitution.

Procedure

- The Base:

  The base begins with the number of components in the principal name (excluding the realm), followed by a colon, and the pattern for building the username from the sections of the principal name. In the pattern section $0 translates to the realm, $1 translates to the first component, and $2 to the second component.

  ```
  [1:$1@$0] translates myusername@APACHE.ORG to myusername@APACHE.ORG
  [2:$1] translates myusername/admin@APACHE.ORG to myusername
  [2:$1%$2] translates myusername/admin@APACHE.ORG to “myusername%admin"
  ```

- The Filter:

  The filter consists of a regular expression (regex) in a parentheses. It must match the generated string for the rule to apply.

  ```
  (.*%admin) matches any string that ends in %admin
  (.*@SOME.DOMAIN) matches any string that ends in @SOME.DOMAIN
  ```

- The Substitution:

  The substitution is a sed rule that translates a regex into a fixed string.

  ```
  s/@ACME\.COM// removes the first instance of @ACME.DOMAIN
  s/@[A-Z]*\.COM// remove the first instance of @ followed by a name
   followed by COM.
  s/X/Y/g replace all of X's in the name with Y
  ```

Example

If your default realm was APACHE.ORG, but you also wanted to take all principals from ACME.COM that had a single component joe@ACME.COM, the following rule would do this:

```
```

Example

To translate names with a second component, you could use these rules:

```
RULE:[1:$1@$0](.@ACME.COM)s/@.//
```

```
RULE:[2:$1@$0](.@ACME.COM)s/@.// DEFAULT
```

Example

To treat all principals from APACHE.ORG with the extension /admin as admin, your rules would look like this:

```
RULE[2:$1%$2@$0](.%admin@APACHE.ORG)s/./admin/
DEFAULT
```

Example

To force username conversion from CaMeLcase or UPPERCASE to lowercase, you could model the following auth_to_local rule examples which have the lowercase switch added:

```
RULE:[1:$1]/L
RULE:[2:$1]/L
RULE:[2:$1;$2](^.*;admin$)s/;admin$///L
RULE:[2:$1;$2](^.*;guest$)s/;guest$//g/L
```

```
RULE:[1:$1]/L
RULE:[2:$1]/L
RULE:[2:$1;$2](^.*;admin$)s/;admin$///L
RULE:[2:$1;$2](^.*;guest$)s/;guest$//g/L
```

And based on these rules, here are the expected output for the following inputs:

"JOE@FOO.COM" to "joe" "Joe/root@FOO.COM" to "joe" "Joe/admin@FOO.COM" to "joe" "Joe/guestguest@FOO.COM" to "joe"

Running the Kerberos Security Wizard

Ambari provides three options for enabling Kerberos: using an existing MIT KDC (Automated Setup), using an existing Active Directory (Automated Setup), or manage Kerberos principals and keytabs manually (Manual Setup).

Automated Setup

When choosing Existing MIT KDC or Existing Active Directory, the Kerberos Wizard prompts for information related to the KDC, the KDC Admin Account and the Service and Ambari principals. Once provided, Ambari will automatically create principals, generate keytabs and distribute keytabs to the hosts in the cluster. The services will be configured for Kerberos and the service components are restarted to authenticate against the KDC. This is the Automated Setup option. See “Launching the Kerberos Wizard (Automated Setup)” for more details.

If you chose to enable Kerberos using the Automated Kerberos Setup option, as part of the enabling Kerberos process, Ambari installs the Kerberos clients on the cluster hosts. Depending on your operating system, the following packages are installed:

Table 1: Packages installed by Ambari for the Kerberos Client

Operating System	Packages
RHEL/CentOS/Oracle Linux 7	krb5-workstation
RHEL/CentOS/Oracle Linux 6	krb5-workstation
SLES 11	krb5-client
Ubuntu/Debian	krb5-user, krb5-config

Manual Setup

When choosing Manage Kerberos principals and keytabs manually, you must create the principals, generate and distribute the keytabs; including you performing the “Ambari Server Kerberos setup”. Ambari will not do this automatically. This is the Manual Setup option. See “Launching the Kerberos Wizard (Manual Setup)” for more details.

Related Information

Launch the Kerberos Wizard (Automated Setup)

Launch the Kerberos Wizard (Manual Setup)

Set Up Kerberos for Ambari Server

Launch the Kerberos Wizard (Automated Setup)

Choose the Kerberos Wizard Automated Setup if you will use an existing MIT KDC or Active Directory, as opposed to managing Kerberos principals and keytabs manually.

Procedure

1. Be sure you have installed and configured your KDC and have prepared the JCE on each host in the cluster.
2. Log in to Ambari Web and Browse to Admin > Kerberos.
3. Click “Enable Kerberos” to launch the wizard.
4. Select the type of KDC you are using and confirm you have met the prerequisites.
5. Provide information about the KDC and admin account.
   a) In the KDC section, enter the following information:
      - In the KDC Host field, the IP address or FQDN for the KDC host. Optionally a port number may be included.
      - In the Realm name field, the default realm to use when creating service principals.
      - (Optional) In the Domains field, provide a list of patterns to use to map hosts in the cluster to the appropriate realm. For example, if your hosts have a common domain in their FQDN such as host1.hortonworks.local and host2.hortonworks.local, you would set this to: .hortonworks.local,hortonworks.local

   b) In the Kadmin section, enter the following information:
      - In the Kadmin Host field, the IP address or FQDN for the KDC administrative host. Optionally a port number may be included.
      - The Admin principal and password that will be used to create principals and keytabs.
      - (Optional) If you have configured Ambari for encrypted passwords, the Save Admin Credentials option will be enabled. With this option, you can have Ambari store the KDC Admin credentials to use when making cluster changes. Refer to “Managing Admin Credentials” for more information on this option.
6. Modify any advanced Kerberos settings based on your environment.
   a) (Optional) To manage your Kerberos client krb5.conf manually (and not have Ambari manage the krb5.conf), expand the Advanced krb5-conf section and uncheck the "Manage" option. You must have the krb5.conf configured on each host.

      When manually managing the krb5.conf it is recommended to ensure that DNS is not used for looking up KDC, and REALM entries. Relying on DNS can cause negative performance, and functional impact. To ensure that DNS is not used, ensure the following entries are set in the libdefaults section of your configuration.

      ```
      [libdefaults]
      dns_lookup_kdc = false
      dns_lookup_realm = false
      ```

b) (Optional) to configure any additional KDC's to be used for this environment, add an entry for each additional KDC to the realms section of the Advanced krb5-conf's krb5-conf template.

```
kdc = {{kdc_host}}
kdc = otherkdc.example.com
```

c) (Optional) To not have Ambari install the Kerberos client libraries on all hosts, expand the Advanced kerberos-env section and uncheck the “Install OS-specific Kerberos client package(s)” option. You must have the Kerberos client utilities installed on each host.

d) (Optional) If your Kerberos client libraries are in non-standard path locations, expand the Advanced kerberos-env section and adjust the “Executable Search Paths” option.

e) (Optional) If your KDC has a password policy, expand the Advanced kerberos-env section and adjust the Password options.

f) (Optional) Ambari will test your Kerberos settings by generating a test principal and authenticating with that principal. To customize the test principal name that Ambari will use, expand the Advanced kerberos-env section and adjust the Test Kerberos Principal value. By default, the test principal name is a combination of cluster name and date (${cluster_name}-${short_date}). This test principal will be deleted after the test is complete.

g) (Optional) If you need to customize the attributes for the principals Ambari will create, when using Active Directory, see “Customizing the Attribute Template” for more information. When using MIT KDC, you can pass Principal Attributes options in the Advanced kerberos-env section. For example, you can set options related to pre-auth or max. renew life by passing:
-requires_preauth -maxrenewlife "7 days"

7. Proceed with the install.
8. Ambari will install Kerberos clients on the hosts and test access to the KDC by testing that Ambari can create a principal, generate a keytab and distribute that keytab.
9. Customize the Kerberos identities used by Hadoop and proceed to kerberize the cluster.

   On the Configure Identities step, be sure to review the principal names, particularly the Ambari Principals on the General tab. These principal names, by default, append the name of the cluster to each of the Ambari principals. You can leave this as default or adjust these by removing the "-${cluster-name}" from principal name string. For example, if your cluster is named HDP and your realm is EXAMPLE.COM, the hdfs principal will be created as hdfs-HDP@EXAMPLE.COM.
10. Confirm your configuration. You can optionally download a CSV file of the principals and keytabs that Ambari will automatically create.
11. Click Next to start the process.
12. After principals have been created and keytabs have been generated and distributed, Ambari updates the cluster configurations, then starts and tests the Services in the cluster.
13. Exit the wizard when complete.
14. Ambari Server communicates with components in the cluster, and now with Kerberos setup, you need to make sure Ambari Server is setup for Kerberos. As part of the automated Kerberos setup process, Ambari Server has been given a keytab and setup is performed. All you need to do is restart Ambari Server for that to take effect. Therefore, restart Ambari Server at this time: ambari-server restart.

Related Information

Install the JCE for Kerberos

Update KDC Admin Credentials

Checklist: Installing and Configuring the KDC

Customizing the Attribute Template

Launch the Kerberos Wizard (Manual Setup)

Choose the Kerberos Wizard Manual Setup if you will manage Kerberos principals and keytabs manually, as opposed to using an existing MIT KDC or Active Directory.

Procedure

1. Be sure you have installed and configured your KDC and have prepared the JCE on each host in the cluster.
2. Log in to Ambari Web and Browse to Admin > Kerberos.
3. Click “Enable Kerberos” to launch the wizard.
4. Select the Manage Kerberos principals and keytabs manually option and confirm you have met the prerequisites.
5. Provide information about the KDC and admin account.

   If your Kerberos client libraries are in non-standard path locations, expand the Advanced kerberos-env section and adjust the “Executable Search Paths” option.
6. Customize the Kerberos identities used by Hadoop and proceed to kerberize the cluster.

   On the Configure Identities step, be sure to review the principal names, particularly the Ambari Principals on the General tab. These principal names, by default, append the name of the cluster to each of the Ambari principals. You can leave this as default or adjust these by removing the "-${cluster-name}" from principal name string. For example, if your cluster is named HDP and your realm is EXAMPLE.COM, the hdfs principal will be created as hdfs-HDP@EXAMPLE.COM.
7. Confirm your configuration. Since you have chosen the Manual Kerberos Setup option, obtain the CSV file for the list of principals and keytabs required for the cluster to work with Kerberos. Do not proceed until you have manually created and distributed the principals and keytabs to the cluster hosts.
8. Click Next to continue.
9. Ambari updates the cluster configurations, then starts and tests the Services in the cluster.
10. Exit the wizard when complete.
11. Finish by completing “Set Up Kerberos for Ambari Server”.

Related Information

Set Up Kerberos for Ambari Server
Install the JCE for Kerberos
Checklist: Installing and Configuring the KDC

Update KDC Admin Credentials

How to update previously-saved KDC credentials in Ambari.

About this task

When you enable Kerberos, if you choose to use an Existing MIT KDC or Existing Active Directory, the Kerberos Wizard prompts for information related to the KDC, the KDC Admin Account credentials, and the Service and Ambari principals. Once provided, Ambari will automatically create principals, generate keytabs and distribute keytabs to the hosts in the cluster. The services will be configured for Kerberos and the service components are restarted to authenticate against the KDC. This is the Kerberos Automated Setup option.

By default, Ambari will not retain the KDC Admin Account credentials you provide unless you have encrypted the passwords stored in Ambari (see “Encrypt Database and LDAP Passwords in Ambari”). If you have not configured Ambari for password encryption, you will be prompted to provide KDC Admin Account credentials whenever cluster changes are made that require KDC principal and/or keytab changes (such as adding services, components and hosts).

If you have configured Ambari for password encryption, you will have an option to Save Admin Credentials. Ambari will use the retained KDC Admin Account credentials to make the KDC changes automatically.

Note:

If you do not have password encryption enabled for Ambari, the Save Admin Credentials option will not be enabled.

Procedure

Updating KDC Credentials:

If you have chosen to Save Admin Credentials when enabling Kerberos, you can update or remove the credentials from Ambari using the following:

a) In Ambari Web, browse to Cluster Admin > Kerberos and click the Manage KDC Credentials button. The Manage KDC Credentials dialog is displayed.
b) If credentials have been previously saved, click Remove to remove the credentials currently stored in Ambari. Once removed, if cluster changes that require KDC principal and/or keytab changes (such as adding services, components and hosts), you will be prompted to enter the KDC Admin Account credentials.
c) Alternatively, to update the KDC Admin Account credentials, enter the Admin principal and password values and click Save.

Related Information

Encrypt Database and LDAP Passwords in Ambari

Customizing the Attribute Template

If you are using the Kerberos Automated setup with Active Directory, depending on your KDC policies, you can customize the attributes that Ambari sets when creating principals.

On the Configure Kerberos step of the wizard, in the Advanced kerberos-env section, you have access to the Ambari Attribute Template. This template (which is based on the Apache Velocity templating syntax) can be modified to adjust which attributes are set on the principals and how those attribute values are derived.

The following table lists the set of computed attribute variables available if you choose to modify the template:

Attribute Variables	Example
$normalized_principal	nn/c6401.ambari.apache.org@EXAMPLE.COM
$principal_name	nn/c6401.ambari.apache.org
$principal_primary	nn
$principal_digest	SHA1 hash of the $normalized_principal
$principal_digest_256	SHA256 hash of the $normalized_principal
$principal_digest_512	SHA512 hash of the $normalized_principal
$principal_instance	c6401.ambari.apache.org
$realm	EXAMPLE.COM
$password	password

Related Information

Apache Velocity

Disable Kerberos Security

After Enabling Kerberos Security, you can disable Kerberos.

Procedure

1. Log in to Ambari Web and Browse to Admin > Kerberos.
2. Click Disable Kerberos to launch the wizard.
3. Complete the wizard.

Related Information

Enabling Kerberos Security

Configuring HDP Components for Kerberos

This section describes how to configure Kerberos for strong authentication for HDP components in an Ambari-managed cluster.

Configuring Kafka for Kerberos

This section describes how to configure Kafka for Kerberos security on an Ambari-managed cluster.

Kerberos security for Kafka is an optional feature. When security is enabled, features include:

- Authentication of client connections (consumer, producer) to brokers
- ACL-based authorization

Kerberos for Kafka Prerequisites

If you are configuring Kafka for Kerberos, your cluster must meet some prerequisites before you can enable Kerberos.

Prerequisite	References*
Ambari-managed cluster with Kafka installed. • Ambari Version 2.1.0.0 or later • Stack version HDP 2.3.2 or later	“Installing, Configuring, and Deploying a Cluster” (link below)
Key Distribution Center (KDC) server installed and running	“Installing and Configuring the KDC” (link below)
JCE installed on all hosts on the cluster (including the Ambari server)	“Enabling Kerberos Authentication Using Ambari” (link below)

Links are for Ambari 2.1.2.0.

When all prerequisites are fulfilled, enable Kerberos security. For more information see “Launching the Kerberos Wizard (Automated Setup)” (link below).

Related Information

Launch the Kerberos Wizard (Automated Setup)

Enabling Kerberos Authentication Using Ambari

Checklist: Installing and Configuring the KDC

Apache Ambari Installation> Installing, Configuring, and Deploying a Cluster

Configuring the Kafka Broker for Kerberos

During the installation process, Ambari configures a series of Kafka settings and creates a JAAS configuration file for the Kafka server.

It is not necessary to modify these settings, but for more information see “Appendix: Kerberos Kafka Configuration Options”.

Related Information

Appendix: Kerberos Kafka Configuration Options

Create Kafka Topics

When you use a script, command, or API to create a topic, an entry is created under ZooKeeper. The only user with access to ZooKeeper is the service account running Kafka (by default, kafka). Therefore, the first step toward creating a Kafka topic on a secure cluster is to run kinit, specifying the Kafka service keytab. The second step is to create the topic.

Procedure

1. Run kinit, specifying the Kafka service keytab. For example:
   kinit -k -t /etc/security/keytabs/kafka.service.keytab kafka/c6401.ambari.apache.org@EXAMPLE.COM
2. Next, create the topic. Run the kafka-topics.sh command-line tool with the following options:

   /bin/kafka-topics.sh --zookeeper <hostname>:<port> --create --topic <topic-name> --partitions <number-of-partitions> --replication-factor <number-of-replicating-servers>

   For more information about kafka-topics.sh parameters, see Basic Kafka Operations on the Apache Kafka website.

   ```
   /bin/kafka-topics.sh --zookeeper c6401.ambari.apache.org:2181 --create --
   topic test_topic --partitions 2 --replication-factor 2

   Created topic "test_topic".
   ```

3. Add permissions:

   By default, permissions are set so that only the Kafka service user has access; no other user can read or write to the new topic. In other words, if your Kafka server is running with principal $KAFKA-USER, only that principal will be able to write to ZooKeeper.

   For information about adding permissions, see “Authorizing Access when Kerberos is Enabled”.

Related Information

Authorizing Access when Kerberos is Enabled

Apache Kafka Documentation

Produce Events or Messages to Kafka on a Secured Cluster

How to produce events/messages to Kafka on a secured cluster.

Before you begin

Make sure that you have enabled access to the topic (via Ranger or native ACLs) for the user associated with the producer process. We recommend that you use Ranger to manage permissions. For more information, see “Apache Ranger User Guide> Adding KAFKA Policies”.

About this task

During the installation process, Ambari configures a series of Kafka client and producer settings, and creates a JAAS configuration file for the Kafka client. It is not necessary to modify these settings, but for more information about them see “Appendix: Kerberos Kafka Configuration Options”.

Note: Only the Kafka Java API is supported for Kerberos. Third-party clients are not supported.

Procedure

1. Specify the path to the JAAS configuration file as one of your JVM parameters:

-Djava.security.auth.login.config=/usr/hdp/current/kafka-broker/config/kafka_client_jaas.conf

For more information about the kafka_client_jaas file, see “JAAS Configuration File for the Kafka Client”.

2. kinit with the principal's keytab.
3. Launch kafka-console-producer.sh with the following configuration options. (Note: these settings are the same as in previous versions, except for the addition of --security-protocol SASL_PLAINTEXT.)

   ./bin/kafka-console-producer.sh --broker-list <hostname:port [,hostname:port, …]> --topic <topic-name> --security-protocol SASL_PLAINTEXT

   ./bin/kafka-console-producer.sh --broker-list c6401.ambari.apache.org:6667,c6402.ambari.apache.org:6667 --topic test_topic --security-protocol SASL_PLAINTEXT

Issue: If you launch the producer from the command-line interface without specifying the security-protocol option, you will see the following error:

```
2015-07-21 04:14:06,611] ERROR fetching topic metadata for topics
[Set(test_topic)] from broker
[ArrayBuffer(BrokerEndPoint(0,c6401.ambari.apache.org,6667),
BrokerEndPoint(1,c6402.ambari.apache.org,6667))] failed
(kafka.utils.CoreUtils$)
kafka.common.KafkaException: fetching topic metadata for topics
[Set(test_topic)] from broker
[ArrayBuffer(BrokerEndPoint(0,c6401.ambari.apache.org,6667),
BrokerEndPoint(1,c6402.ambari.apache.org,6667))] failed
     at kafka.client.ClientUtils$.fetchTopicMetadata(ClientUtils.scala:73)
Caused by: java.io.EOFException: Received -1 when reading from channel,
 socket has likely been closed.
     at kafka.utils.CoreUtils$.read(CoreUtils.scala:193)
     at
 kafka.network.BoundedByteBufferReceive.readFrom(BoundedByteBufferReceive.scala:54)
```

Solution: Add --security-protocol SASL_PLAINTEXT to the kafka-console-producer.sh runtime options.

Producer Code Example for a Kerberos-Enabled Cluster

The following example shows sample code for a producer in a Kerberos-enabled Kafka cluster. Note that the SECURITY_PROTOCOL_CONFIG property is set to SASL_PLAINTEXT.

```
package com.hortonworks.example.kafka.producer;

import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

import java.util.Properties;
import java.util.Random;

public class BasicProducerExample {

   public static void main(String[] args){

       Properties props = new Properties();
       props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
 "kafka.example.com:6667");

       // specify the protocol for SSL Encryption
       props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG,
 "SASL_PLAINTEXT");
```

```
        props.put(ProducerConfig.ACKS_CONFIG, "all");
        props.put(ProducerConfig.RETRIES_CONFIG, 0);
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
 "org.apache.kafka.common.serialization.StringSerializer");
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
 "org.apache.kafka.common.serialization.StringSerializer");

        Producer<String, String> producer = new KafkaProducer<String,
 String>(props);
        TestCallback callback = new TestCallback();
        Random rnd = new Random();
        for (long i = 0; i < 100 ; i++) {
            ProducerRecord<String, String> data = new ProducerRecord<String,
 String>(
                    "test-topic", "key-" + i, "message-"+i );
            producer.send(data, callback);
        }

        producer.close();
    }


    private static class TestCallback implements Callback {
        @Override
        public void onCompletion(RecordMetadata recordMetadata, Exception e)
 {
            if (e != null) {
                System.out.println("Error while producing message to topic :"
 + recordMetadata);
                e.printStackTrace();
            } else {
                String message = String.format("sent message
 to topic:%s partition:%s  offset:%s", recordMetadata.topic(),
 recordMetadata.partition(), recordMetadata.offset());
                System.out.println(message);
            }
        }
    }

}
```

To run the example, issue the following command:

```
$ java -Djava.security.auth.login.config=/usr/hdp/
current/kafka-broker/config/kafka_client_jaas.conf
 com.hortonworks.example.kafka.producer.BasicProducerExample
```

Related Information

Appendix: Kerberos Kafka Configuration Options

JAAS Configuration File for the Kafka Server

Apache Ranger User Guide> Adding KAFKA Policies

Consume Events or Messages from Kafka on a Secured Cluster

How to consume events/messages from Kafka on a secured cluster.

Before you begin

Make sure that you have enabled access to the topic (via Ranger or native ACLs) for the user associated with the consumer process. We recommend that you use Ranger to manage permissions. For more information, see “Apache Ranger User Guide> Adding KAFKA Policies”.

About this task

During the installation process, Ambari configures a series of Kafka client and producer settings, and creates a JAAS configuration file for the Kafka client. It is not necessary to modify these values, but for more information see "Appendix: Kerberos Kafka Configuration Options".

Note: Only the Kafka Java API is supported for Kerberos. Third-party clients are not supported.

Procedure

1. Specify the path to the JAAS configuration file as one of your JVM parameters.

   For more information about the kafka_client_jaas file, see "JAAS Configuration File for the Kafka Client".

   -Djava.security.auth.login.config=/usr/hdp/current/kafka-broker/config/kafka_client_jaas.conf
2. kinit with the principal's keytab.
3. Launch kafka-console-consumer.sh with the following configuration settings. (Note: these settings are the same as in previous versions, except for the addition of --security-protocol SASL_PLAINTEXT.)

   ./bin/kafka-console-consumer.sh --zookeeper c6401.ambari.apache.org:2181 --topic test_topic --from-beginning --security-protocol SASL_PLAINTEXT

Issue: If you launch the consumer from the command-line interface without specifying the security-protocol option, you will see the following error:

```
2015-07-21 04:14:06,611] ERROR fetching topic metadata for topics
[Set(test_topic)] from broker
[ArrayBuffer(BrokerEndPoint(0,c6401.ambari.apache.org,6667),
BrokerEndPoint(1,c6402.ambari.apache.org,6667))] failed
(kafka.utils.CoreUtils$)
kafka.common.KafkaException: fetching topic metadata for topics
[Set(test_topic)] from broker
[ArrayBuffer(BrokerEndPoint(0,c6401.ambari.apache.org,6667),
BrokerEndPoint(1,c6402.ambari.apache.org,6667))] failed
     at kafka.client.ClientUtils$.fetchTopicMetadata(ClientUtils.scala:73)
Caused by: java.io.EOFException: Received -1 when reading from channel,
 socket has likely been closed.
     at kafka.utils.CoreUtils$.read(CoreUtils.scala:193)
     at
 kafka.network.BoundedByteBufferReceive.readFrom(BoundedByteBufferReceive.scala:54)
```

Solution: Add --security-protocol SASL_PLAINTEXT to the kafka-console-consumer.sh runtime options.

Consumer Code Example for a Kerberos-Enabled Cluster

The following example shows sample code for a producer in a Kerberos-enabled Kafka cluster. Note that the SECURITY_PROTOCOL_CONFIG property is set to SASL_PLAINTEXT.

```
package com.hortonworks.example.kafka.consumer;

import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.TopicPartition;

import java.util.Collection;
import java.util.Collections;
import java.util.Properties;
```

```
public class BasicConsumerExample {

   public static void main(String[] args) {

       Properties consumerConfig = new Properties();
       consumerConfig.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
 "kafka.example.com:6667");

       // specify the protocol for SSL Encryption
       consumerConfig.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG,
 "SASL_PLAINTEXT");

       consumerConfig.put(ConsumerConfig.GROUP_ID_CONFIG, "my-group");
       consumerConfig.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
 "earliest");
       consumerConfig.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
 "org.apache.kafka.common.serialization.StringDeserializer");
       consumerConfig.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
 "org.apache.kafka.common.serialization.StringDeserializer");
       KafkaConsumer<byte[], byte[]> consumer = new
 KafkaConsumer<>(consumerConfig);
       TestConsumerRebalanceListener rebalanceListener = new
 TestConsumerRebalanceListener();
       consumer.subscribe(Collections.singletonList("test-topic"),
 rebalanceListener);

       while (true) {
           ConsumerRecords<byte[], byte[]> records = consumer.poll(1000);
           for (ConsumerRecord<byte[], byte[]> record : records) {
               System.out.printf("Received Message topic =%s, partition =%s,
 offset = %d, key = %s, value = %s\n", record.topic(), record.partition(),
 record.offset(), record.key(), record.value());
           }

           consumer.commitSync();
       }

   }

   private static class  TestConsumerRebalanceListener implements
 ConsumerRebalanceListener {
       @Override
       public void onPartitionsRevoked(Collection<TopicPartition>
 partitions) {
           System.out.println("Called onPartitionsRevoked with partitions:"
 + partitions);
       }

       @Override
       public void onPartitionsAssigned(Collection<TopicPartition>
 partitions) {
           System.out.println("Called onPartitionsAssigned with partitions:"
 + partitions);
       }
   }

}
```

To run the example, issue the following command:

```
# java -Djava.security.auth.login.config=/usr/hdp/
current/kafka-broker/config/kafka_client_jaas.conf
 com.hortonworks.example.kafka.consumer.BasicConsumerExample
```

Related Information

Appendix: Kerberos Kafka Configuration Options
JAAS Configuration File for the Kafka Server
Apache Ranger User Guide> Adding KAFKA Policies

Authorizing Access when Kerberos is Enabled

Kafka ships with a pluggable Authorizer and an out-of-box authorizer implementation that uses ZooKeeper to store Access Control Lists (ACLs).

Authorization can be done via Ranger (see the “Kafka” section of the Ranger Install Guide) or with native ACLs.

A Kafka ACL entry has the following general format:

Principal P is [Allowed/Denied] Operation O From Host H On Resource R

where

- A principal is any entity that can be authenticated by the system, such as a user account, a thread or process running in the security context of a user account, or security groups of such accounts. Principal is specified in the PrincipalType:PrincipalName (user:dev@EXAMPLE.COM) format. Specify user:* to indicate all principals.

  Principal is a comma-separated list of principals. Specify * to indicate all principals. (A principal is any entity that can be authenticated by the system, such as a user account, a thread or process running in the security context of a user account, or security groups of such accounts.)
- Operation can be one of: READ, WRITE, CREATE, DESCRIBE, or ALL.
- Resource is a topic name, a consumer group name, or the string “kafka-cluster” to indicate a cluster-level resource (only used with a CREATE operation).
- Host is the client host IP address. Specify * to indicate all hosts.

Appendix: Kerberos Kafka Configuration Options

Reference material for Kerberos Kafka configuration options.

Server.properties key value pairs

Ambari configures the following Kafka values during the installation process. Settings are stored as key-value pairs stored in an underlying server.properties configuration file.

listeners

A comma-separated list of URIs that Kafka will listen on, and their protocols.

Required property with three parts:

<protocol>:<hostname>:<port>

Set <protocol> to SASL_PLAINTEXT, to specify the protocol that server accepts connections. SASL authentication will be used over a plaintext channel. Once SASL authentication is established between client and server, the session will have the client’s principal as an authenticated user. The broker can only accept SASL (Kerberos) connections, and there is no wire encryption applied. (Note: For a non-secure cluster, <protocol> should be set to PLAINTEXT.)

Set hostname to the hostname associated with the node you are installing. Kerberos uses this value and "principal" to construct the Kerberos service name. Specify hostname 0.0.0.0 to bind to all interfaces. Leave hostname empty to bind to the default interface.

Set port to the Kafka service port. When Kafka is installed using Ambari, the default port number is 6667.

Examples of legal listener lists::

listeners=SASL_PLAINTEXT://kafka1.host1.com:6667

listeners=PLAINTEXT://myhost:9092, TRACE://:9091, SASL_PLAINTEXT://0.0.0.0:9093

advertised.listeners

A list of listeners to publish to ZooKeeper for clients to use, if different than the listeners specified in the preceding section.

In IaaS environments, this value might need to be different from the interface to which the broker binds.

If advertised.listeners is not set, the value for listeners will be used.

Required value with three parts:

<protocol>:<hostname>:<port>

Set protocol to SASL_PLAINTEXT, to specify the protocol that server accepts connections. SASL authentication will be used over a plaintext channel. Once SASL authentication is established between client and server, the session will have the client's principal as an authenticated user. The broker can only accept SASL (Kerberos) connections, and there is no wire encryption applied. (Note: For a non-secure cluster, <protocol> should be set to PLAINTEXT.)

Set hostname to the hostname associated with the node you are installing. Kerberos uses this and "principal" to construct the Kerberos service name.

Set port to the Kafka service port. When Kafka is installed using Ambari, the default port number is 6667.

For example:

advertised.listeners=SASL_PLAINTEXT://kafka1.host1.com:6667

security.inter.broker.protocol

Specifies the inter-broker communication protocol. In a Kerberized cluster, brokers are required to communicate over SASL. (This approach supports replication of topic data.) Set the value to SASL_PLAINTEXT:

security.inter.broker.protocol=SASL_PLAINTEXT

authorizer.class.name

Configures the authorizer class.

Set this value to kafka.security.auth.SimpleAclAuthorizer:

authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer

For more information, see "Authorizing Access when Kerberos is Enabled."

principal.to.local.class

Transforms Kerberos principals to their local Unix usernames.

Set this value to kafka.security.auth.KerberosPrincipalToLocal:

principal.to.local.class=kafka.security.auth.KerberosPrincipalToLocal

super.users

Specifies a list of user accounts that will have all cluster permissions. By default, these super users have all permissions that would otherwise need to be added through the kafka-acls.sh script. Note, however, that their permissions do not include the ability to create topics through kafka-topics.sh, as this involves direct interaction with ZooKeeper.

Set this value to a list of user:<account> pairs separated by semicolons. Note that Ambari adds user:kafka when Kerberos is enabled.

Here is an example:

```
super.users=user:bob;user:alice
```

JAAS Configuration File for the Kafka Server

The Java Authentication and Authorization Service (JAAS) API supplies user authentication and authorization services for Java applications.

After enabling Kerberos, Ambari sets up a JAAS login configuration file for the Kafka server. This file is used to authenticate the Kafka broker against Kerberos. The file is stored at:

/usr/hdp/current/kafka-broker/config/kafka_server_jaas.conf

Ambari adds the following settings to the file. (Note: serviceName="kafka" is required for connections from other brokers.)

```
KafkaServer {
       com.sun.security.auth.module.Krb5LoginModule required
       useKeyTab=true
       keyTab="/etc/security/keytabs/kafka.service.keytab"
       storeKey=true
       useTicketCache=false
       serviceName="kafka"
       principal="kafka/c6401.ambari.apache.org@EXAMPLE.COM";
       };

Client { // used for zookeeper connection
       com.sun.security.auth.module.Krb5LoginModule required
       useKeyTab=true
       keyTab="/etc/security/keytabs/kafka.service.keytab"
       storeKey=true
       useTicketCache=false
       serviceName="zookeeper"
       principal="kafka/c6401.ambari.apache.org@EXAMPLE.COM";
       };
```

Configuration Setting for the Kafka Producer

Reference information for the configuration setting for the Kafka producer.

After enabling Kerberos, Ambari sets the following key-value pair in the server.properties file:

security.protocol=SASL_PLAINTEXT

JAAS Configuration File for the Kafka Client

After enabling Kerberos, Ambari sets up a JAAS login configuration file for the Kafka client. Settings in this file will be used for any client (consumer, producer) that connects to a Kerberos-enabled Kafka cluster.

The file is stored at:

/usr/hdp/current/kafka-broker/config/kafka_client_jaas.conf

Ambari adds the following settings to the file. (Note: serviceName=kafka is required for connections from other brokers.)

Kafka client configuration with keytab, for producers:

```
KafkaClient {
     com.sun.security.auth.module.Krb5LoginModule required
     useKeyTab=true
     keyTab="/etc/security/keytabs/storm.service.keytab"
     storeKey=true
     useTicketCache=false
     serviceName="kafka"
     principal="storm@EXAMPLE.COM";
    };
```

Kafka client configuration without keytab, for producers:

```
KafkaClient {
     com.sun.security.auth.module.Krb5LoginModule required
     useTicketCache=true
     renewTicket=true
     serviceName="kafka";
    };
```

Kafka client configuration for consumers:

```
KafkaClient {
     com.sun.security.auth.module.Krb5LoginModule required
     useTicketCache=true
     renewTicket=true
     serviceName="kafka";
     };
```

Configuring Storm for Kerberos

This section describes how to configure Storm for Kerberos security on an Ambari-managed cluster.

Kerberos for Storm Prerequisites

If you are configuring Storm for Kerberos, your cluster must meet some prerequisites before you can enable Kerberos.

Note: Links point to Ambari version 2.2.1.0. If your cluster runs a different version of Ambari, refer to the Ambari document for your version of software.

Before you enable Kerberos, your cluster must meet the following prerequisites. (Note:)

Prerequisite	References
Ambari-managed cluster with Storm installed and running. • Ambari Version 2.2.1.0 or later • Stack version HDP 2.4.0 or later	“Installing, Configuring, and Deploying a Cluster” (link below)
Key Distribution Center (KDC) server installed and running.	“Installing and Configuring the KDC” (link below)
JCE installed on all hosts on the cluster (including the Ambari server).	“Enabling Kerberos Authentication Using Ambari” (link below)

When all prerequisites are fulfilled, enable Kerberos security. For more information, see “Launching the Kerberos Wizard (Automated Setup)”.

Related Information

Enabling Kerberos Authentication Using Ambari

Launch the Kerberos Wizard (Automated Setup)

Checklist: Installing and Configuring the KDC

Apache Ambari Installation> Installing, Configuring, and Deploying a Cluster

Designating a Storm Client Node

At this point in the configuration process there is no notion of a Storm client node (you won’t be able to select “client” via Ambari). There are two choices when specifying a Storm client node.

At this point in the configuration process there is no notion of a Storm client node (you won’t be able to select “client” via Ambari).

To specify a Storm client node, choose one of the following two approaches, described in the following subsections:

- Dedicate or use an existing independent gateway node as a storm client
- Use one of your existing storm nodes (such as nimbus, supervisors, or drpc) as a client. Choose this option if you prefer not to add a gateway node for Storm.

Dedicate or Use an Existing Gateway Node

How to designate a storm client node using an existing gateway node (edge node).

Procedure

1. Install the storm package on the node: sudo yum install storm_<version>.
   For HDP 2.4: sudo yum install storm_2_4*
2. Create a file at /etc/storm/conf/client_jaas.conf, and add the following entry to it:

```
StormClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useTicketCache=true
    renewTicket=true
    serviceName="nimbus"
    };
```

3. Add the following settings to the /etc/storm/conf/storm.yaml configuration file:

```
nimbus.seeds: <nimbus-host-array>
nimbus.thrift.port: 6627
java.security.auth.login.config: "/etc/storm/conf/client_jaas.conf"
storm.thrift.transport:
 "org.apache.storm.security.auth.kerberos.KerberosSaslTransportPlugin"
```

   where <nimbus-host-array> is an array of hostnames running Nimbus. (The value should come from /etc/storm/conf/storm.yaml.)

   nimbus.seeds: ["c6401.ambari.apache.org", "c6402.ambari.apache.org"]

Use an Existing Storm Node

How to designate a storm client node using an existing Storm node (edge node).

About this task

To use one of your existing Storm nodes (such as nimbus, supervisors, or drpc) as a Storm client node, complete the following steps for every user who requires Storm access (for example, to run Storm commands or deploy topologies):

Procedure

1. Create a .storm directory in the user's home directory. For example, user john should have a directory called /home/john/.storm/.
2. Add the following settings to the /etc/storm/conf/storm.yaml configuration file:

```
nimbus.seeds: <nimbus-host-array>
nimbus.thrift.port: 6627
java.security.auth.login.config: "/etc/storm/conf/client_jaas.conf"
storm.thrift.transport:
 "org.apache.storm.security.auth.kerberos.KerberosSaslTransportPlugin"
```

   where <nimbus-host-array> is an array of hostnames running Nimbus (the value should come from /etc/storm/conf/storm.yaml).

   nimbus.seeds: ["c6401.ambari.apache.org", "c6402.ambari.apache.org"]

What to do next

Repeat these steps for every user who requires Storm access.

Running Storm Commands

After configuring the client/gateway node, run kinit (with the principal's keytab) before issuing Storm commands.

Running Workers as Users

In Storm secure mode, workers can run as the user (owner of the topology) who deployed the topology. This topic describes how to enable this.

Procedure

1. Make sure all users who are going to deploy topologies have a UNIX account on all of the Storm nodes. Workers will run under the UNIX account for topologies deployed by the user.

   For user testuser1 and principal testuser1/c6401.ambari.apache.org, make sure there is a corresponding testuser1 UNIX account.
2. Add the following configuration under "Custom storm-site" in the Ambari Storm configuration screen: supervisor.run.worker.as.user : true.
3. Restart Storm components.

Accessing the Storm UI

How to access the Storm UI.

About this task

The Storm UI uses SPNEGO AUTH when in Kerberos mode.

Procedure

1. Before accessing the UI, configure your browser for SPNEGO authorization, as shown in the following table:

   Table 2: Browser Settings for Storm UI

Browser	Configuration
Safari	No changes needed.
Firefox	**a.** Go to about:config and search for network.negotiate-auth.trusted-uris. **b.** Double-click and add the following value: "http://storm-ui-hostname:ui-port" **c.** Replace the storm-ui-hostname value with the hostname where your UI is running. **d.** Replace the ui-port value with the Storm UI port.
Chrome	From the command line, issue: google-chrome --auth-server-whitelist="<storm-ui-hostname>" --auth-negotiate-delegate-whitelist="<storm-ui-hostname>"
Internet Explorer	**a.** Configure trusted websites to include "storm-ui-hostname". **b.** Allow negotiation for the UI website.

2. Then kinit before accessing the Storm UI.

Accessing the Storm UI Active Directory Trust Configuration

How to access the Storm UI AD trust configuration.

About this task

If your cluster is configured with Active Directory Trust, use the Active Directory ticket to communicate with MIT KDC for secure negotiation. Here are the additional configuration steps:

Procedure

1. Make sure UI Kerberos authentication-to-local rules are configured properly. Once a principal from Active Directory is used for negotiation with MIT KDC, you need a rule to translate it to the local account on the Storm UI node. Many times those can be copied from core-site.xml.

```
ui.filter.params:
   "type": "kerberos"
   "kerberos.principal": "HTTP/nimbus.host1.com"
   "kerberos.keytab": "/vagrant/keytabs/http.keytab"
   "kerberos.name.rules": "RULE:[2:$1@$0]([jt]t@.*EXAMPLE.COM)s/.*/
$MAPRED_USER/ RULE:[2:$1@$0]([nd]n@.*EXAMPLE.COM)s/.*/$HDFS_USER/DEFAULT"
```

   Note: Rules are listed as strings, and are not separated by commas.

2. Create mappings for MIT domain KDC and associated resources used for the domain, in this case Storm UI.

   On a Windows workstation, you would run the following commands from the command line:

   ksetup /AddKDC $DOMAIN $KDC

   ksetup /AddHostToRealmMap $hadoop_resource $Domain

   Note: this step adds registry entries in HKLM\System\CurrentControlSet\Control\Lsa\Kerberos\HostToRealm.

To troubleshoot configuration issues, try accessing the Storm UI within the cluster using the curl command.

For example:

curl -i --negotiate -u:anyUser -b ~/cookiejar.txt -c ~/cookiejar.txt http://storm-ui-hostname:8080/api/v1/cluster/summary

This will help you determine whether the Kerberos UI configuration is working.

To isolate the issue, use Storm service keytabs and user principals.

Two other important things to check are:

- Make sure that the trust is working properly.
- Make sure that the encryption types match on both KDCs.

Kerberos Storm Security Properties

A reference table that lists important Storm security properties.

Configuration Property	Description	Example
nimbus.authorizer	This is a pluggable authorizer for a Storm Nimbus node. SimpleACLAuthorizer is the default implementation. Note: Admins can also grant permissions via the Ranger authorizer UI. For more information, see the Ranger User's Guide.	"org.apache.storm.security.auth.authorizer.SimpleACLAuthorizer

nimbus.admins	Add Nimbus admin users. These users will have super user permissions on all topologies deployed, and will be able to perform other admin operations (such as rebalance, activate, deactivate and kill), even if they are not the owners of the topology. By default, only users who deployed the topologies have access to admin operations such as rebalance, activate, deactivate, and kill.	"John" "Abc"
topology.users:	This and the following config can be added as part of the topology file. The users listed in this setting will have owner privileges for the specified topology.	`Config conf = new Config() conf.put(”topology.users”,Lists.ne StormSubmitter.submitTopology(topo conf, builder.createTopology());`
topology.groups	Similar to topology.users. Use this to add group-level permissions to a topology.	`Config conf = new Config() conf.put(”topology.groups”,Lists.n StormSubmitter.submitTopology(topo conf, builder.createTopology());`

Known Issues with Storm for Kerberos

Reference of known issues with Storm for Kerberos.

Issue: Ambari does not show the security configuration on the Storm configuration tab, so you cannot add users to nimbus.admins.

Workaround: To give permissions to other users, use topology.users or topology.groups.

Issue: In AD+MIT setup, when trying to access Nimbus on a Kerberized cluster a HTTP 413 full HEAD error is received. (STORM-633)

Workaround: Add ui.header.buffer.bytes : “65536” under “Custom storm-site" on the Ambari Storm configuration tab.

Issue: Log viewer. We recommend against creating HTTP principal keytabs for supervisors. This can cause the SPNEGO protocol to fail.

Workaround:

1. Add the HTTP principal for Storm supervisor nodes too. For example:

   sudo /usr/sbin/kadmin.local -q 'addprinc -randkey HTTP/<supervisor-hostname>

   where

   <supervisor-hostname> is your hostname and domain for Kerberos; for example: supervisor1.host1.com@HOST1.COM
2. Add this principal for all hosts that run supervisor machines.

   For example:

   sudo /usr/sbin/kadmin.local -q "ktadd -k /etc/security/keytabs/spnego.service.keytab HTTP/ supervisor1.host1.com@HOST1.COM"
3. Add the newly created HTTP principals to the spnego.service.keytab file.
4. Make sure that the spnego.service.keytab file has “storm” user privileges for read operations.
5. Distribute this keytab to all supervisor hosts.

6. On the supervisor node, edit /etc/storm/conf/storm.yaml. Change the ui.filter.parameters as follows, replacing <supervisor-hostname> with the hostname of your supervisor process:

   "type": "kerberos"

   "kerberos.principal": "HTTP/<supervisor-hostname>"

   "kerberos.keytab": "/vagrant/keytabs/http.keytab"
7. On each supervisor machine change the Kerberos.principal hostname to that supervisor's hostname.
8. Restart the log viewer.
9. Add supervisor hosts to network.negotiate-auth.trusted-uris (similar to the steps needed to access the Storm UI).

Related Information

STORM-633

Securing Apache HBase in a production environment

HBase is a popular distributed key-value store modeled after Google's BigTable. HBase can support extremely fast lookups, high write throughput and strong consistency making it suitable for a wide variety of use cases ranging from application data store like Facebook Messaging to analytical use cases like Yahoo Flurry. HBase stores data on HDFS, providing it linear scaling as well as fault-tolerance.

Similar to HDFS, Kerberos integration works by adding SASL based authentication layer in HBase protocol requiring valid SPNs for authentication. In addition, HBase itself uses Kerberos to authenticate against HDFS while storing data. HBase supports cell-level access control, providing a very granular authorization layer.

Installing Apache HBase with Kerberos on an existing HDP cluster

You can install Apache HBase with Kerberos enabled on an HDP cluster.

Before you begin

- You must have an operational HDP cluster with Kerberos enabled.
- You must have Kerberos admin access.

Follow these steps to install HBase with Kerberos enabled:

Procedure

1. Log in to Ambari .
2. From the Actions menu, click Add Service.
3. From the services list, select HBase and click Next.
4. Select the location to install HBase Master.
5. Select the nodes to install HBase Region Servers.
6. Review the configuration details and modify it based on your performance tuning needs and then click Next.

   You can customize the services later as well.
7. Review the Kerberos Service Principal Name (SPNs) that will be created for HBase deployment and click Next.
8. Review the configuration details and click Deploy.

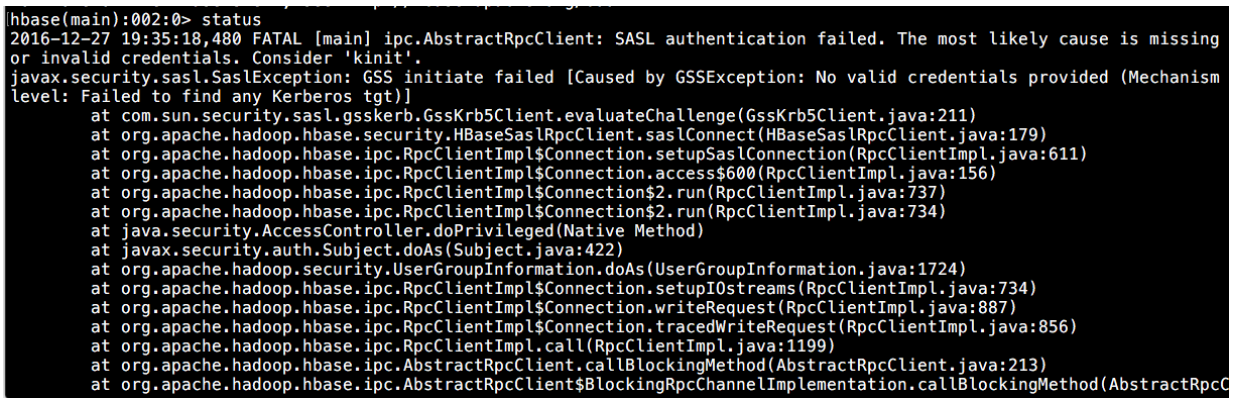
   If Kerberos admin credentials were not stored while enabling Kerberos on HDFS, Ambari will prompt you for the credentials again.
9. Click Save once credentials are entered.
10. Wait for the installation to complete.
11. Review any errors encountered during installation and click Next.

Verify if kerberos is enabled for HBase

Follow these steps to verify, if kerberos is enabled for Apache HBase:

Procedure

1. Log in to the Ambari node where HBase client is installed.
2. Start the HBase shell.
3. On the HBase Master host machine, execute the status command:



In this example, the command fails with a stack trace, because there is no TGT. therefore can’t authenticate to HBase using Kerberos.

4. Get a Ticket Granting Ticket (TGT).

Here are the examples that shows the input and the output for creating a local TGT when you run a kinit operation. In the first example, you run a `kinit` on the command line first and then run the application. The second example automatically obtains a TGT through a keytab.

Examplen 1 of secure client:

```
package com.hortonworks.hbase.examples;

import java.io.IOException;
import java.util.Objects;
```

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Admin;
import org.apache.hadoop.hbase.client.ColumnFamilyDescriptorBuilder;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.Table;
import org.apache.hadoop.hbase.client.TableDescriptorBuilder;
import org.apache.hadoop.hbase.util.Bytes;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Write and read data from HBase, expects HBASE_CONF_DIR and
 HADOOP_CONF_DIR on the classpath and a valid Kerberos
 * ticket in a ticket cache (e.g. kinit).
 */
public class ExampleSecureClient implements Runnable {
  private static final Logger LOG =
 LoggerFactory.getLogger(ExampleSecureClient.class);
  private static final TableName TABLE_NAME =
 TableName.valueOf("example_secure_client");
  private static final byte[] CF = Bytes.toBytes("f1");

  private final Configuration conf;

  public ExampleSecureClient(Configuration conf) {
    this.conf = Objects.requireNonNull(conf);
  }

  @Override public void run() {
    try (Connection conn = ConnectionFactory.createConnection(conf)) {
      writeAndRead(conn, TABLE_NAME, CF);
      LOG.info("Success!");
    } catch (Exception e) {
      LOG.error("Uncaught exception running example", e);
      throw new RuntimeException(e);
    }
  }

  void writeAndRead(Connection conn, TableName tn, byte[] family) throws
 IOException {
    final Admin admin = conn.getAdmin();

    // Delete the table if it already exists
    if (admin.tableExists(tn)) {
      admin.disableTable(tn);
      admin.deleteTable(tn);
    }

    // Create our table

 admin.createTable(TableDescriptorBuilder.newBuilder(tn).setColumnFamily(
        ColumnFamilyDescriptorBuilder.of(family)).build());

    final Table table = conn.getTable(tn);
    Put p = new Put(Bytes.toBytes("row1"));
    p.addColumn(family, Bytes.toBytes("q1"), Bytes.toBytes("value"));
    LOG.info("Writing update: row1 -> value");
```

```
    table.put(p);

    Result r = table.get(new Get(Bytes.toBytes("row1")));
    assert r.size() == 1;
    LOG.info("Read row1: {}", r);
  }

  public static void main(String[] args) {
    final Configuration conf = HBaseConfiguration.create();
    new ExampleSecureClient(conf).run();
  }
}
```

Example 1 of Ticket Cache output:

```
2018-06-12 13:44:40,144 WARN  [main] util.NativeCodeLoader: Unable to load
 native-hadoop library for your platform... using builtin-java classes
 where applicable
2018-06-12 13:44:40,975 INFO  [main] zookeeper.ReadOnlyZKClient: Connect
 0x62e136d3 to my.fqdn:2181 with session timeout=90000ms, retries 6, retry
 interval 1000ms, keepAlive=60000ms
2018-06-12 13:44:42,806 INFO  [main] client.HBaseAdmin: Started disable of
 example_secure_client
2018-06-12 13:44:44,159 INFO  [main] client.HBaseAdmin: Operation:
 DISABLE, Table Name: default:example_secure_client completed
2018-06-12 13:44:44,590 INFO  [main] client.HBaseAdmin: Operation: DELETE,
 Table Name: default:example_secure_client completed
2018-06-12 13:44:46,040 INFO  [main] client.HBaseAdmin: Operation: CREATE,
 Table Name: default:example_secure_client completed
2018-06-12 13:44:46,041 INFO  [main] examples.ExampleSecureClient: Writing
 update: row1 -> value
2018-06-12 13:44:46,183 INFO  [main] examples.ExampleSecureClient: Read
 row1: keyvalues={row1/f1:q1/1528825486175/Put/vlen=5/seqid=0}
2018-06-12 13:44:46,183 INFO  [main] examples.ExampleSecureClient:
 Success!
2018-06-12 13:44:46,183 INFO  [main] client.ConnectionImplementation:
 Closing master protocol: MasterService
2018-06-12 13:44:46,183 INFO  [main] zookeeper.ReadOnlyZKClient: Close
 zookeeper connection 0x62e136d3 to my.fqdn:2181
```

5. Execute the status command again by logging in through principal and keytab.

   Example 2 of secure client with keytab login:

```
package com.hortonworks.hbase.examples;

import java.io.File;
import java.security.PrivilegedAction;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.security.UserGroupInformation;

public class ExampleSecureClientWithKeytabLogin {

  public static void main(String[] args) throws Exception {
    final Configuration conf = HBaseConfiguration.create();

    final String principal = "myself@EXAMPLE.COM";
    final File keytab = new File("/etc/security/keytabs/myself.keytab");
```

```
    assert keytab.isFile() : "Provided keytab '" + keytab + "' is not a
 regular file.";

    UserGroupInformation.setConfiguration(conf);
    UserGroupInformation ugi =
 UserGroupInformation.loginUserFromKeytabAndReturnUGI(
        principal, keytab.getAbsolutePath());

    ugi.doAs(new PrivilegedAction<Void>() {
      @Override public Void run() {
        new ExampleSecureClient(conf).run();
        return null;
      }
    });
  }
}
```

The following example shows the output resulting from the keytab login:

```
2018-06-12 13:29:23,057 WARN  [main] util.NativeCodeLoader: Unable to load
 native-hadoop library for your platform... using builtin-java classes
 where applicable
2018-06-12 13:29:23,574 INFO  [main] zookeeper.ReadOnlyZKClient: Connect
 0x192d43ce to my.fqdn:2181 with session timeout=90000ms, retries 6, retry
 interval 1000ms, keepAlive=60000ms
2018-06-12 13:29:29,172 INFO  [main] client.HBaseAdmin: Started disable of
 example_secure_client
2018-06-12 13:29:30,456 INFO  [main] client.HBaseAdmin: Operation:
 DISABLE, Table Name: default:example_secure_client completed
2018-06-12 13:29:30,702 INFO  [main] client.HBaseAdmin: Operation: DELETE,
 Table Name: default:example_secure_client completed
2018-06-12 13:29:33,005 INFO  [main] client.HBaseAdmin: Operation: CREATE,
 Table Name: default:example_secure_client completed
2018-06-12 13:29:33,006 INFO  [main] examples.ExampleSecureClient: Writing
 update: row1 -> value
2018-06-12 13:29:33,071 INFO  [main] examples.ExampleSecureClient: Read
 row1: keyvalues={row1/f1:q1/1528824573066/Put/vlen=5/seqid=0}
2018-06-12 13:29:33,071 INFO  [main] examples.ExampleSecureClient:
 Success!
2018-06-12 13:29:33,071 INFO  [main] client.ConnectionImplementation:
 Closing master protocol: MasterService
2018-06-12 13:29:33,071 INFO  [main] zookeeper.ReadOnlyZKClient: Close
 zookeeper connection 0x192d43ce to my.fqdn:2181
```

Access Kerberos-enabled HBase cluster using a Java client

You can access Kerberos-enabled HBase cluster using a Java client.

Before you begin

- HDP cluster with Kerberos enabled.
- You are working in a Java 8, Maven 3 and Eclipse development environment.
- You have administrator access to Kerberos KDC.

Perform the following tasks to connect to HBase using a Java client and perform a simple Put operation to a table.

Procedure

1. “Download configurations”
2. “Set up client account”
3. “Access Kerberos-Enabled HBase cluster using a Java client”

Related Information

Download configurations
Set up client account
Create the Java client

Download configurations

Follow these steps to download the required configurations:

Procedure

1. From Ambari, extract the HBase and HDFS files to conf directory, which will save all the configuration details.

   These files must be extracted under the $HBASE_CONF_DIR directory, where $HBASE_CONF_DIR is the directory to store the HBase configuration files. For example, /etc/hbase/conf.
2. From KDC, download the krb5.conf file from /etc/krb5.con. You can also place the configuration snippets in the directory.

```
includedir /etc/krb5.conf.d/

              [logging]
              default = FILE:/var/log/krb5libs.log
              kdc = FILE:/var/log/krb5kdc.log
              admin_server = FILE:/var/log/kadmind.log

              [libdefaults]
              dns_lookup_realm = false
              ticket_lifetime = 24h
              renew_lifetime = 7d
              forwardable = true
              rdns = false
              default_realm = HWFIELD.COM
              default_ccache_name = KEYRING:persistent:%{uid}

             [realms]
             HWFIELD.COM = {
             kdc = ambud-hdp-3.field.hortonworks.com
             admin_server = ambud-hdp-3.field.hortonworks.com
             }

            [domain_realm]
            .hwfield.com = HWFIELD.COM
           hwfield.com = HWFIELD.COM
```

Set up client account

Follow these steps to provision a kerberos account for the client and grant permissions to that account in HBase, so that you can create, read and write tables.

Procedure

1. Log in to KDC.
2. Switch to root directory.
3. Run kadmin.local:

```
$ sudo kadmin.local
kadmin.local: addprinc myself
WARNING: no policy specified for myself@EXAMPLE.COM; defaulting to no
 policy
Enter password for principal "myself@EXAMPLE.COM":
Re-enter password for principal "myself@EXAMPLE.COM":
```

```
Principal "myself@EXAMPLE.COM" created.
kadmin.local:  xst -k /etc/security/keytabs/myself.keytab -norandkey
 myself
Entry for principal myself with kvno 1, encryption type aes256-cts-hmac-
sha1-96 added to keytab
WRFILE:/etc/security/keytabs/myself.keytab.
Entry for principal myself with kvno 1, encryption type aes128-cts-hmac-
sha1-96 added to keytab
WRFILE:/etc/security/keytabs/myself.keytab.
```

4. Copy the keytab file to the conf directory.
5. Grant permissions in HBase. For more information, See Configure HBase for Access Control Lists (ACL).

```
klist -k /etc/security/keytabs/hbase.headless.keytab
```

Optional step: You should secure the keytab file so that only the HBase process has access to the keytab. This can be accomplished by running a command.

```
$>sudo chmod 700 /etc/security/keytabs/hbase.headless.keytab
```

```
$ kinit -kt /etc/security/keytabs/hbase.headless.keytab hbase
$ hbase shell
hbase(main):001:0> status
1 active master, 0 backup masters, 4 servers, 1 dead, 1.2500 average load
```

6. Authorize admin permissions to the user. You can also customize this to restrict this account for minimal access. For more information see, http://hbase.apache.org/0.94/book/hbase.accesscontrol.configuration.html#d1984e4744

Example

hbase(main):001:0> grant 'myself', 'C'

Create the Java client

Follow these steps to create a Java client:

Procedure

1. Launch Eclipse.
2. Create a simple Maven project.
3. Add the hbase-client and hadoop-auth dependencies.

   The client uses the Hadoop UGI utility class to perform a Kerberos authentication using the keytab file. It sets up the context so that all operations are performed under the hbase-user2 security context. Then, it performs the required HBase operations, namely check / create table and perform a put and get operations.

```
<dependencies>
    <dependency>
      <groupId>org.apache.hbase</groupId>
      <artifactId>hbase-client</artifactId>
      <version>${hbase.version}</version>
      <exclusions>
        <exclusion>
          <groupId>org.apache.hadoop</groupId>
          <artifactId>hadoop-aws</artifactId>
        </exclusion>
      </exclusions>
```

```
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-auth</artifactId>
      <version>${hadoop.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-common</artifactId>
      <version>${hadoop.version}</version>
    </dependency>
    <dependency>
```

4. Execute the HBase Java client code from a node that can be reverse-DNS resolved.

   This is part of Kerberos authentication. Therefore, running it from a machine that does not share the same DNS infrastructure as the HDP cluster results in authentication failure.

5. To validate that your Kerberos authentication / keytab / principal does indeed work, you can also perform a simple Kerberos authentication from Java. This provides you with some insight into how Java JAAS and Kerberos works.

   It is highly recommended that you use either Maven Shade Plugin or Maven Jar Plugin to automatically package dependencies into a fat-client JAR. You can also use Eclipse Export feature, however this is not recommended for production code base.