Installing Kerberos on Hadoop cluster

Hadoop Security

Authorisation vs Authentication

One of the more confusing topics in Hadoop is how authorization and authentication work in the system. The first and most important thing to recognize is the subtle, yet extremely important, differentiation between authorization and authentication, so let’s define these terms first:

*Authentication* is the process of determining whether someone is who they claim to be.

*Authorization* is the function of specifying access rights to resources.

In simpler terms, authentication is a way of proving who I am, and authorization is a way of determining what I can do.

<https://blog.cloudera.com/blog/2012/03/authorization-and-authentication-in-hadoop/>

*Kerberos is a protocol for*[*authenticating*](https://searchsecurity.techtarget.com/definition/authentication)*service requests between trusted hosts across an untrusted network, such as the internet. Kerberos is built in to all major operating systems, including Microsoft Windows, Apple OS X, FreeBSD and Linux.*

*Kerberos protocol overview*

*Kerberos and Hadoop At a high level, there are three steps that a client must take to access a service when using Kerberos, each of which involves a message exchange with a server: 1. Authentication. The client authenticates itself to the Authentication Server and receives a timestamped Ticket-Granting Ticket (TGT).*

*2. Authorization. The client uses the TGT to request a service ticket from the TicketGranting Server.*

*3. Service request. The client uses the service ticket to authenticate itself to the server that is providing the service the client is using. In the case of Hadoop, this might be*

*The authorization and service request steps are not user-level actions; the client performs these steps on the user’s behalf. The authentication step, however, is normally carried out explicitly by the user using the kinit command, which will prompt for a password. However, this doesn’t mean you need to enter your password every time you run a job or access HDFS, since TGTs last for 10 hours by default (and can be renewed for up to a week). It’s common to automate authentication at operating system login time, thereby providing single sign-on to Hadoop. In cases where you don’t want to be prompted for a password (for running an unattended MapReduce job, for example), you can create a Kerberos keytab file using the ktutil command. A keytab is a file that stores passwords and may be supplied to kinit with the -t option.*

*An example Let’s look at an example of the process in action. The first step is to enable Kerberos authentication by setting the hadoop.security.authentication property in core-site.xml to kerberos. [74] The default setting is simple, which signifies that the old backwardcompatible (but insecure) behavior of using the operating system username to determine*

*identity should be employed. We also need to enable service-level authorization by setting hadoop.security.authorization to true in the same file. You may configure access control lists (ACLs) in the hadoop-policy.xml configuration file to control which users and groups have permission to connect to each Hadoop service. Services are defined at the protocol level, so there are ones for MapReduce job submission, namenode communication, and so on. By default, all ACLs are set to \*, which means that all users have permission to access each service; however, on a real cluster you should lock the ACLs down to only those users and groups that should have access. The format for an ACL is a comma-separated list of usernames, followed by whitespace, followed by a comma-separated list of group names. For example, the ACL preston,howard directors,inventors would authorize access to users named preston or howard, or in groups directors or inventors. With Kerberos authentication turned on, let’s see what happens when we try to copy a local file to HDFS: % hadoop fs -put quangle.txt . 10/07/03 15:44:58 WARN ipc.Client: Exception encountered while connecting to the server: javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSException: No valid credentials provided (Mechanism level: Failed to find any Kerberos tgt)] Bad connection to FS. command aborted. exception: Call to localhost/ 127.0.0.1:8020 failed on local exception: java.io.IOException: javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSException: No valid credentials provided (Mechanism level: Failed to find any Kerberos tgt)] The operation fails because we don’t have a Kerberos ticket. We can get one by authenticating to the KDC, using kinit: % kinit Password for hadoop-user@LOCALDOMAIN: password % hadoop fs -put quangle.txt . % hadoop fs -stat %n quangle.txt quangle.txt*

*And we see that the file is successfully written to HDFS. Notice that even though we carried out two filesystem commands, we only needed to call kinit once, since the Kerberos ticket is valid for 10 hours (use the klist command to see the expiry time of your tickets and kdestroy to invalidate your tickets). After we get a ticket, everything works just as it normally would.*

*Delegation Tokens*

*In a distributed system such as HDFS or MapReduce, there are many client-server interactions, each of which must be authenticated. For example, an HDFS read operation will involve multiple calls to the namenode and calls to one or more datanodes. Instead of using the three-step Kerberos ticket exchange protocol to authenticate each call, which would present a high load on the KDC on a busy cluster, Hadoop uses delegation tokens to allow later authenticated access without having to contact the KDC again. Delegation tokens are created and used transparently by Hadoop on behalf of users, so there’s no action you need to take as a user beyond using kinit to sign in, but it’s useful to have a basic idea of how they are used.*

*Other Security Enhancements Security has been tightened throughout the Hadoop stack to protect against unauthorized access to resources. The more notable features are listed here: Tasks can be run using the operating system account for the user who submitted the job, rather than the user running the node manager. This means that the operating system is used to isolate running tasks, so they can’t send signals to each other (to kill another user’s tasks, for example) and so local information, such as task data, is kept private via local filesystem permissions. This feature is enabled by setting yarn.nodemanager.container-executor.class to org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor. [75] In addition, administrators need to ensure that each user is given an account on every node in the cluster (typically using LDAP). When tasks are run as the user who submitted the job, the distributed cache (see Distributed Cache) is secure. Files that are world-readable are put in a shared cache (the insecure default); otherwise, they go in a private cache, readable only by the owner. Users can view and modify only their own jobs, not others. This is enabled by setting mapreduce.cluster.acls.enabled to true. There are two job configuration*

*Steps for installing Kerberos server*

1. *Sudo* yum install krb5-server krb5-libs krb5-workstation
2. *Vi /etc/krb5.conf*

*default\_realm = VIGDATOS.COM*

*default\_ccache\_name = KEYRING:persistent:%{uid}*

*[realms]*

*VIGDATOS.COM = {*

*kdc = c1*

*admin\_server = c1*

*}*

*[domain\_realm]*

*.vigdatos.com = VIGDATOS.COM*

*vigdatos.com = VIGDATOS.COM*

1. *Create database with realm name*

*'/var/kerberos/krb5kdc/principal'*

*kdb5\_util create -s -r VIGDATOS.COM*

1. Start kdc server and kdc admin server

[hadoop@c1 etc]$ systemctl start krb5kdc kadmin

[hadoop@c1 etc]$ systemctl enable krb5kdc kadmin

1. Install all Kerberos on all machines

sudo yum install krb5-workstation krb5-libs

1. Move krb5.conf to all servers from kdc server
2. sudo mv /tmp/krb5.conf /etc/krb5.conf
3. create principal

[hadoop@c1 ~]$ sudo /usr/sbin/kadmin.local -q "addprinc root/admin"

[sudo] password for hadoop:

Authenticating as principal root/admin@VIGDATOS.COM with password.

WARNING: no policy specified for root/admin@VIGDATOS.COM; defaulting to no policy

Enter password for principal "root/admin@VIGDATOS.COM":

Re-enter password for principal "root/admin@VIGDATOS.COM":

Principal "root/admin@VIGDATOS.COM" created.

kadmin.local

addprinc root/admin

kinit root/admin

1. cat /var/kerberos/krb5kdc/kadm5.acl
2. sudo vi /var/kerberos/krb5kdc/kadm5.acl

change example to vigdatos

1. Install all Kerberos clients on your machine by using the command below:

    # yum install krb5-server krb5-libs krb5-auth-dialog krb5-workstation

1. On all server sudo kdb5\_util create -s
2. On all servers check sudo kadmin -p root/admin

kadmin.local: addprinc -randkey hadoop/hadoop

WARNING: no policy specified for hadoop/hadoop@VIGDATOS.COM; defaulting to no policy

Principal "hadoop/hadoop@VIGDATOS.COM" created.

kadmin.local: ktadd -k /home/hadoop/hadoop.keytab hadoop/hadoop

Entry for principal hadoop/hadoop with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab WRFILE:/home/hadoop/hadoop.keytab.

Entry for principal hadoop/hadoop with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keytab WRFILE:/home/hadoop/hadoop.keytab.

Entry for principal hadoop/hadoop with kvno 2, encryption type des3-cbc-sha1 added to keytab WRFILE:/home/hadoop/hadoop.keytab.

Entry for principal hadoop/hadoop with kvno 2, encryption type arcfour-hmac added to keytab WRFILE:/home/hadoop/hadoop.keytab.

Entry for principal hadoop/hadoop with kvno 2, encryption type camellia256-cts-cmac added to keytab WRFILE:/home/hadoop/hadoop.keytab.

Entry for principal hadoop/hadoop with kvno 2, encryption type camellia128-cts-cmac added to keytab WRFILE:/home/hadoop/hadoop.keytab.

Entry for principal hadoop/hadoop with kvno 2, encryption type des-hmac-sha1 added to keytab WRFILE:/home/hadoop/hadoop.keytab.

Entry for principal hadoop/hadoop with kvno 2, encryption type des-cbc-md5 added to keytab WRFILE:/home/hadoop/hadoop.keytab.

1. kinit -p root/admin

<https://bluedata.zendesk.com/hc/en-us/articles/115007578207-How-To-Set-Up-an-MIT-KDC-Server-for-a-Hadoop-Cluster>

sudo scp hadoop.keytab c3:/tmp

sudo mv /tmp/hadoop.keytab /home/hadoop/

1. sudo kinit -k -t /home/hadoop/hadoop.keytab hadoop/Hadoop
2. <https://docs.oracle.com/cd/E19683-01/806-4078/6jd6cjs1a/index.html>
3. <https://docs.oracle.com/cd/E19683-01/806-4078/6jd6cjs1b/index.html>
4. <http://www.hadoopadmin.co.in/kerberos-2/>

<https://blog.godatadriven.com/kerberos-cloudera-setup>

<https://gist.github.com/ashrithr/4767927948eca70845db>

install ntp

sudo yum -y install ntp

sudo vi /var/kerberos/krb5kdc/kadm5.acl

[hadoop@c2 ~]$ klist

Ticket cache: KEYRING:persistent:1001:1001

Default principal: root/admin@VIGDATOS.COM

Valid starting Expires Service principal

07/15/2019 20:07:59 07/16/2019 20:07:59 krbtgt/VIGDATOS.COM@VIGDATOS.COM

[hadoop@c2 ~]$

**1.2 What is a Kerberos Principal?**

A Kerberos *principal* is a unique identity to which Kerberos can assign tickets. Principals can have an arbitrary number of components. Each component is separated by a component separator, generally `/'. The last component is the realm, separated from the rest of the principal by the realm separator, generally `@'. If there is no realm component in the principal, then it will be assumed that the principal is in the default realm for the context in which it is being used.

Traditionally, a principal is divided into three parts: the *primary*, the *instance*, and the *realm*. The format of a typical Kerberos V5 principal is primary/instance@REALM.

* The *primary* is the first part of the principal. In the case of a user, it's the same as your username. For a host, the primary is the word host.
* The *instance* is an optional string that qualifies the primary. The instance is separated from the primary by a slash (/). In the case of a user, the instance is usually null, but a user might also have an additional principal, with an instance called admin, which he/she uses to administrate a database. The principal jennifer@ATHENA.MIT.EDU is completely separate from the principal jennifer/admin@ATHENA.MIT.EDU, with a separate password, and separate permissions. In the case of a host, the instance is the fully qualified hostname, e.g., daffodil.mit.edu.
* The *realm* is your Kerberos realm. In most cases, your Kerberos realm is your domain name, in upper-case letters. For example, the machine daffodil.example.com would be in the realm EXAMPLE.COM.

A **Kerberos realm** is the domain over which a **Kerberos** authentication server has the authority to authenticate a user, host or service. A **realm** name is often, but not always the upper case version of the name of the DNS domain over which it presides

<https://www.certdepot.net/rhel7-use-kerberos-control-access-nfs-network-shares/>