Kerberos入门与实战

一、 Kerberos简介

二、 Kerberos体系结构

三、 Kerberos工作机制

四、 Kerberos安装部署

五、 CDH启用kerberos

六、 Kerberos开发使用

Kerberos简介-Kerberos是什么？

Kerberos定义

Kerberos（/ˈkərbərəs/)是一种计算机网络授权协议，用来在非安全网络中，对个人通信以安全的手段进行身份认证。这个词又指麻省理工学院为这个协议开发的一套计算机软件。软件设计上采用客户端/服务器结构，并且能够进行相互认证，即客户端和服务器端均可对对方进行身份认证。可以用于防止窃听、防止重放攻击、保护数据完整性等场合，是一种应用对称密钥体制进行密钥管理的系统。Kerberos的扩展产品也使用公开密钥加密方法进行认证。

当有N个人使用该系统时，为确保在任意两个人之间进行秘密对话，系统至少保存有它与每个人的共享密钥，所需的最少会话密钥数为N个。

Kerberos网络认证协议允许某实体在非安全网络环境下通信，向另一个实体以一种安全的方式证明自己的身份。它也指由麻省理工实现此协议，并发布的一套免费软件。它的设计主要针对客户-服务器模型，并提供了一系列交互认证——用户和服务器(或具体服务)都能验证对方的身份。Kerberos协议可以保护网络实体免受窃听和重复攻击。

Kerberos协议基于对称密码学，并需要一个值得信赖的第三方。Kerberos协议的扩展可以为认证的某些阶段提供公钥密码学支持。

Kerberos简介-Kerberos的历史由来

Kerberos是MIT开发的，用来保护雅典娜工程（Project Athena）提供的网络服务器。麻省理工在版权许可的情况下，制作一个Kerberos的免费实现工具，这种情况类似于BSD。在2007年，麻省理工组成一个Kerberos协会，以此推动Kerberos的持续发展。

在2005年，互联网工程任务组（IETF）Kerberos工作小组更新了规范，更新包括

"Kerberos 5加密和校验和规范"。

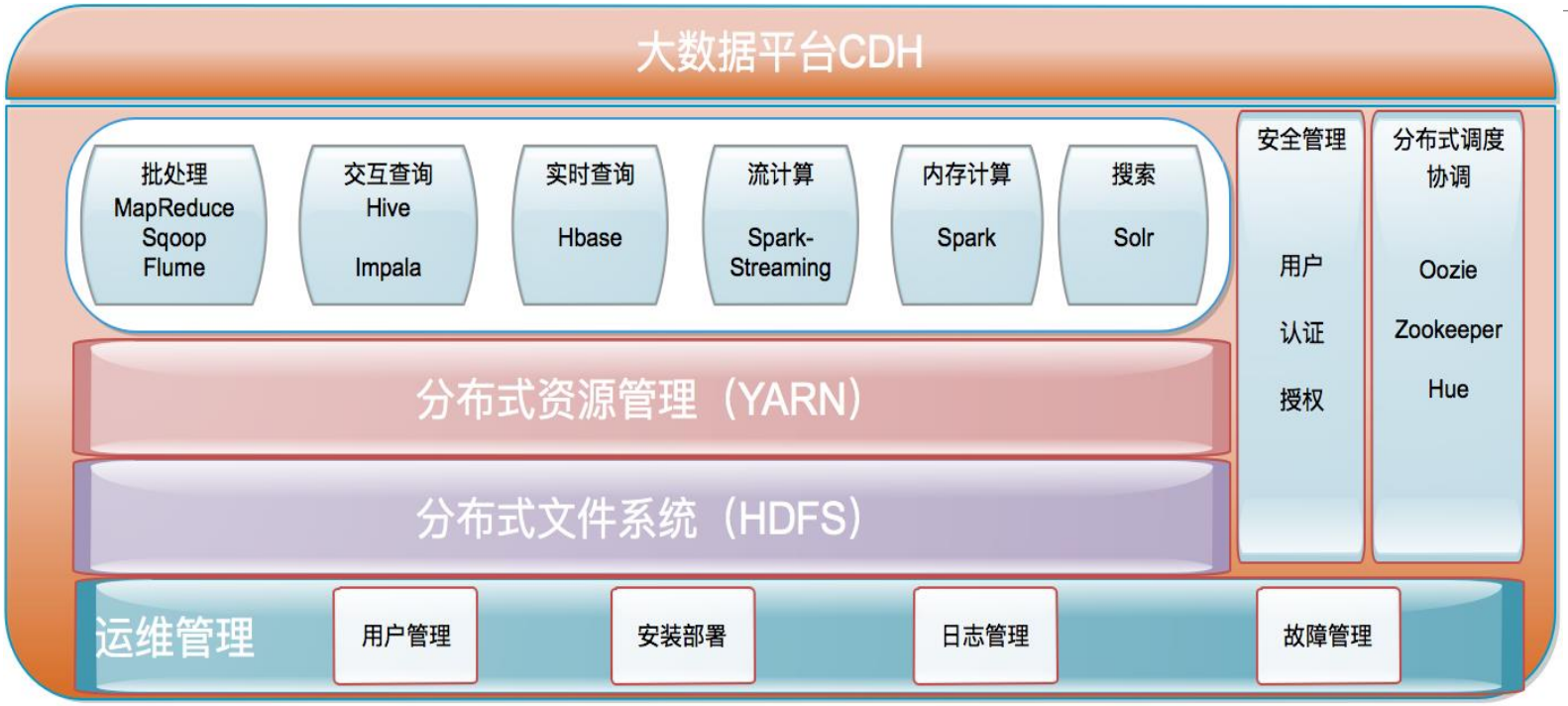
"Kerberos 5高级加密标准（AES）加密"。

" Kerberos网络认证服务（版本5）”—Kerberos版本5规范的新版本。这个版 本用更细化和明确的解释说明协议的一些细节和使用方法。

"Kerberos 5通用安全服务应用程序接口（GSS-API）机制：版本2"—通用安全服务应用程序接口（GSS-API）规范的新版本。

目前[illegible]的Kerberos，版本目前为5。

Kerberos体系结构-Kerberos在Hadoop中的位置



Kerberos体系结构-Kerberos在Hadoop中概念

- **Kerberos Principal**
  - 主体：在Kerberos中用户或者服务被成为主体，一般由两部分或者三部分组成：primary主要标识、instance实例名称（可选）、realm领域。在Kerberos的认证系统中通过主体标识唯一身份，在CDH中主体一般是操作系统用户名或者服务名。
  - username@YOUR-REALM.COM　　app1@HADOOP.COM
  - name/fully.qualified.domain.name@YOUR-REALM.COM
    hbase/node1@HADOOP.COM
  - Kerberos通过将票据Tickets分配给kerberos主体使其可以访问启用Kerberos集群的Hadoop服务。

- Kerberos Keytab
  - Keytab文件包含了Kerberos主体和该主体秘钥的加密副本，对于Hadoop守护进程来说每个keytab文件是唯一的，因为实例包含了主机名，如hbase/node@HADOOP.COM，该文件用来认证kerberos主机上的主体，因此keytab一定要安全保存。

- Delegation Tokens
  - 用户通过kerberos认证后，提交作业，如果用户此时登出系统，则后续的认证通过委托令牌的方式完成，委托令牌和Nn共享秘钥，用于模拟用户来获得执行任务，委托令牌有效期为一天，可以通过更新程序更新令牌不超过7天，任务完成以后委托令牌取消。

Kerberos工作机制-相关概念

Kerberos依靠Ticket的概念来工作，每个部分概念如下：

- KDC（Key Distribution Center）= 密钥分发中心
  由AS和TGS组成，是所有通信的主要枢纽，保存有每个客户端或者服务的秘钥副本，辅助完成通信认证。
- AS（Authentication Server）= 认证服务器
- TGS（Ticket Granting Server）= 票据授权服务器
- TGT（Ticket Granting Ticket）= 票据授权票据，票据的票据
- SS（Service Server）= 特定服务提供端

Kerberos涉及的参与者：

- 请求访问某个资源的主体，可以是用户或者服务。
- 被请求的资源，一般是具体某个服务，比如hive等。
- KDC

Kerberos工作机制-功能概述

要开启一个认证会话，客户端首先将用户名发送到KDC的AS进行认证(一般是通过kinit命令完成)，KDC服务器生成相应的票据授权票据(TGT)并打上时间戳，TGT是一个用于请求和其他服务通信的票据，并在数据库中查找该请求用户的密码，并用查找到的密码对TGT进行加密，将加密结果返回给请求用户。

客户端收到返回结果，使用自己的密码解密得到TGT票据授权票据，该TGT会在一段时间后自动失效，有些程序可以用户登录期间进行自动更新，比如hadoop的hdfs用户。当客户端需要请求服务时，客户端将该TGT发送到KDC的TGS服务，当用户的TGT通过验证并且有权限访问所申请的服务时，TGS生成一个被请求服务对应的Ticket和和Session Key，并发给请求客户端。

客户端将该Ticket和要请求的服务一同发送给目的服务端，完成验证并获得相应服务。

简单地说，用户先用共享密钥从KDC的AS认证服务器得到一个身份证明。随后，用户使用这个身份证明与SS通信，而不使用共享密钥。

Kerberos工作机制-工作原理

- **客户端认证过程 -------客户端(Client)从认证服务器(AS)获取票据的票据（TGT）**

① 客户端向AS发送明文信息，申请基于该用户的请求服务，比如‘用户APP1想请求服务’。（注意：用户不向AS发送“用户密钥”(user's secret key)，也不发送密码）该AS能够从本地数据库中查询到该申请用户的密码，并通过相同途径转换成相同的“用户密钥”(user's secret key)。

② AS检查请求用户ID是否存在于Kerberos数据库中，如果存在则通过验证，返回如下信息。

- **消息A：会话秘钥，Client/TGS会话密钥(Client/TGS Session Key)**（该Session Key用在将来Client与TGS的通信（会话）上），通过用户密钥(user ‘s secret key)进行加密。
- **消息B：票据授权票据(TGT)**（TGT包括： “Client/TGS会话密钥”(Client/TGS Session Key)，KDC名字，用户ID，IP地址，TGT有效期），通过KDC中**TGS密钥(TGS ‘s secret key)**进行加密。
- KDC将响应结果返回，包括加密后的新会话秘钥、TGT。

③ Client收到返回消息后，Client用自己的密钥解密返回的加密会话密钥，从而得到解密后的会话密钥**Client/TGS会话密钥(Client/TGS Session Key)**，（注意：Client不能解密TGT，因为TGT是用TGS密钥(TGS's secret key)加密的）。拥有了“Client/TGS会话密钥”(Client/TGS Session Key)，Client就足以通过TGS进行认证了。

Kerberos工作机制-工作原理

- **服务授权 -------（client从TGS获取票据(client-to-server ticket)）**

① 当客户端需要请求某个服务时，发送如下信息到KDC。

- **消息C：**1.获取的返回**TGT，消息B**，即通过**TGS密钥(TGS‘s secret key)**进行加密的TGT。2.想要获取服务的服务ID。
- **消息D：认证元组**（用户ID、IP地址、时间戳），通过**Client/TGS会话密钥(Client/TGS Session Key)** 加密。

② KDC收到请求后，TGS检查KDC数据库中是否存在该服务ID，如果存在，则TGS用自己的**TGS密钥(TGS‘s secret key)**解密请求消息获得**TGT**，得到之前生成的**Client/TGS会话密钥(Client/TGS Session Key)**。TGS在用该会话密钥**Client/TGS Session Key**解密认证元组，得到（用户ID、IP地址、时间戳），并验证TGT和认证元组，如果验证通过返回如下:

- **消息E：Client-Server票据(Client-To-Server Ticket)，**（该Ticket包括：**Client/SS会话密钥(Client/Server Session Key)**，用户ID，用户网址，有效期），通过提供该服务的**服务密钥(service's secret key)**进行加密。
- **消息F：Client/SS会话密钥(Client/Server Session Key)，**（该Session Key用在将来Client与Server Service的通信（会话）上），通过**Client/TGS会话密钥(Client/TGS Session Key)**进行加密。

③ 客户端收到响应后通过**Client/TGS会话密钥”(Client/TGS Session Key)**解密消息得到**Client/SS会话密钥(Client/Server Session Key)。**（注意：Client不能解密**Client-Server票据(Client-To-Server Ticket)**，因为是用“服务密钥”(service's secret key)加密的）。

Kerberos工作机制-工作原理

- **服务请求** ------- Client从SS获取服务

① 通过获取的Client/SS会话密钥”(Client/Server Session Key)后，Client就可以使用服务器提供的服务，Client向服务器发送如下信息。

- **消息E：Client-Server票据(Client-To-Server Ticket)** ，（该Ticket包括：**Client/SS会话密钥(Client/Server Session Key)** ，用户ID，用户网址，有效期），通过提供该服务的**服务密钥(service's secret key)**进行加密。
- **消息G：新认证元组**（用户ID、IP地址、时间戳），通过**Client/SS会话密钥(Client/Server Session Key)** 进行加密。

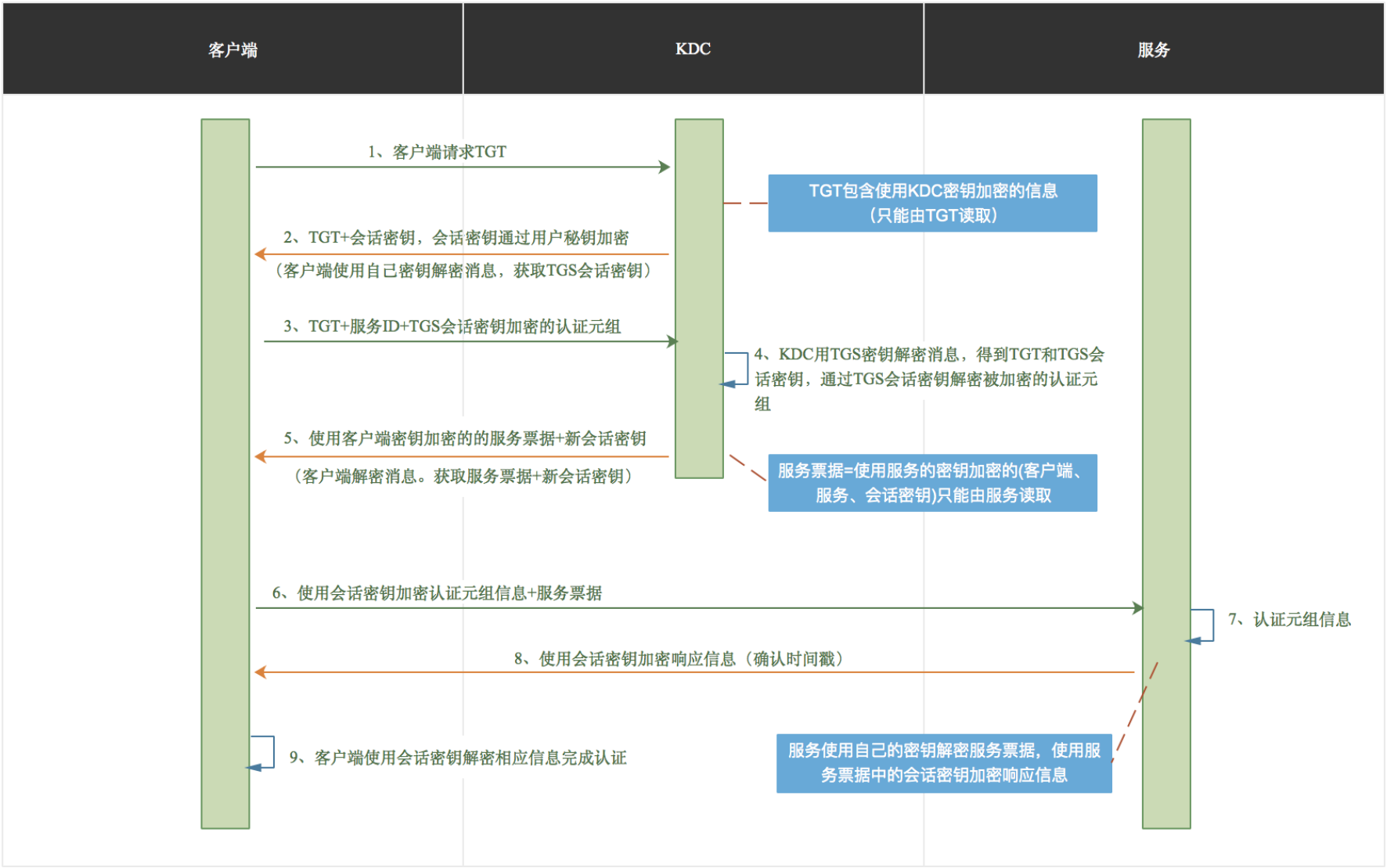
② SS用自己的密钥(service ‘s secret key)解密消息**Client-Server票据(Client-To-Server Ticket)**从而得到TGS提供的Client/SS会话密钥(Client/Server Session Key)。再用这个会话密钥解密加密的**新认证元组**得到**新认证元组**。（同TGS一样）对Ticket和认证元组进行验证，验证通过则返回1条消息（确认函：确证身份真实，乐于提供服务）：

- **消息H：新时间戳**，（新时间戳是：Client发送的时间戳加1，v5已经取消这一做法），通过**Client/SS会话密钥(Client/Server Session Key)**进行加密。

③ Client通过**Client/SS会话密钥(Client/Server Session Key)**解密消息**H**，得到新时间戳并验证其是否正确。验证通过的话则客户端可以信赖服务器，并向服务器（SS）发送服务请求。

④ 服务器（SS）向客户端提供相应的服务。

Kerberos工作机制-工作原理



Kerberos安装部署

安装和配置

① 根据需要卸载存在旧版本krb，检查系统是否自带了krb: `rpm -qa | grep krb5`

② 如果存在旧版krb则卸载: `rpm -e --nodeps +文件名(包括krb5-lib,krb5-workstation)`

- kdc主机需要安装krb5-lib-1.13.2,krb5-server-1.13.2,krb5-workstation-1.13.2
- kdc的从节点上只需安装krb5-lib-1.1.3.2,krb5-workstation-1.13.2
- Kdc节点安装过程：

```
[root@node3 krb5kdc]# rpm -e --nodeps krb5-workstation-1.12.2-14.el7.x86_64
[root@node3 krb5kdc]# rpm -qa|grep krb5
sssd-krb5-common-1.12.2-58.el7.x86_64
sssd-krb5-1.12.2-58.el7.x86_64
pam_krb5-2.4.8-4.el7.x86_64
[root@node3 krb5kdc]# cd /root
[root@node3 ~]# ls
anaconda-ks.cfg  krb5-1.15.tar.gz                  krb5-server-1.13.2-10.el7.x86_64.rpm       ntp-dev-4.3.93
krb5-1.15        krb5-libs-1.13.2-10.el7.x86_64.rpm  krb5-workstation-1.13.2-10.el7.x86_64.rpm  ntp-dev-4.3.93.tar.gz
[root@node3 ~]# clear
[root@node3 ~]# ls
anaconda-ks.cfg  krb5-1.15.tar.gz                  krb5-server-1.13.2-10.el7.x86_64.rpm       ntp-dev-4.3.93
krb5-1.15        krb5-libs-1.13.2-10.el7.x86_64.rpm  krb5-workstation-1.13.2-10.el7.x86_64.rpm  ntp-dev-4.3.93.tar.gz
[root@node3 ~]# rpm -ivh krb5-libs-1.13.2-10.el7.x86_64.rpm
Preparing...                          ################################# [100%]
Updating / installing...
   1:krb5-libs-1.13.2-10.el7          ################################# [100%]
[root@node3 ~]# rpm -ivh krb5-workstation-1.13.2-10.el7.x86_64.rpm
Preparing...                          ################################# [100%]
Updating / installing...
   1:krb5-workstation-1.13.2-10.el7   ################################# [100%]
[root@node3 ~]# rpm -ivh krb5-server-1.13.2-10.el7.x86_64.rpm
Preparing...                          ################################# [100%]
Updating / installing...
   1:krb5-server-1.13.2-10.el7        ################################# [100%]
```

- Kdc从节点同上，不安装krb5-server-1.13.2

③ 修改配置文件

- vi /var/kerberos/krb5kdc/kdc.conf(只有kdc主机上修改)

Kerberos安装部署

```
[kdcdefaults]
kdc_ports = 88
kdc_tcp_ports = 88

[realms]
HADOOP.COM = {
#master_key_type = aes256-cts
acl_file = /var/kerberos/krb5kdc/kadm5.acl
dict_file = /usr/share/dict/words
admin_keytab = /var/kerberos/krb5kdc/kadm5.keytab
max_renewable_life = 7d
supported_enctypes = aes128-cts:normal des3-hmac-sha1:normal arcfour-hmac:normal des-hma
}
```

④ 配置文件说明

- HADOOP.COM:是设定的realms。名字随意。大小写敏感一般为了识别使用全部大写。
- supported_enctypes:支持的校验方式。注意把aes256-cts去掉。

⑤ 修改所有节点的Kerberos配置信息(包含kdc主机和从节点机器)

Kerberos安装部署

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

[libdefaults]
default_realm = HADOOP.COM
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
# udp_preference_limit = 1

[realms]
HADOOP.COM = {
kdc = vmw201
admin_server = vmw201
}

[domain_realm]
.hadoop.com = HADOOP.COM
hadoop.com = HADOOP.COM
```

Kerberos安装部署

Kerberos初始化---------以下操作均为在Kdc主机上操作

① 创建/初始化kerberos database

- 执行/usr/sbin/kdb5_util create -s -r HADOOP.COM
- 可能遇到的问题：重建数据库，步骤如下两步。
- 1、 kdb5_util -r HADOOP.COM -m destroy -f
- 2、将目录/var/kerberos/krb5kdc下的principal相关的文件删除即

② 当kerberos database创建好以后，查看/var/kerberos/krb5kdc 目录，文件如下

```
1 kadm5.acl
2 kdc.conf
3 principal
4 principal.kadm5
5 principal.kadm5.lock
6 principal.ok
```

③ 添加管理员

- Kdc主机上执行 /usr/sbin/kadmin.local -q “addprinc admin/admin”，并为其设置密码。

Kerberos安装部署

④ 为database admin设置ACL权限

- 修改acl文件来设置权，该acl文件的默认路径是 /var/kerberos/krb5kdc/kadm5.acl（也可以在文件kdc.conf中修改）
- 将文件/var/kerberos/krb5kdc/kadm5.acl的内容编辑为 */admin@HADOOP.COM
- 代表名称匹配*/admin@HADOOP.COM 都认为是admin，权限是 *。代表全部权限

⑤ 在KDC主机启动kerberos后台进程

- 启动kdc　　service krb5kdc start
- 启动kadmin　　service kadmin start

Kerberos安装部署

验证

① Kdc验证。创建用户，在kdc主机执行　kadmin.lcoal 回车

- kadmin.local: addprinc app1　添加主体/用户 app1
- kadmin.local: listprincs　获取主体或者用户列表
- Exit退出
- 执行kinit app1 输入密码
- 执行klist　查看输出信息

② 从节点验证

- 执行 kinit app1　回车 输入密码
- 执行klist显示输出信息

③ 生成keytab文件

- 执行 kadmin.local:xst -k /XXX/app1.keytab app1　在指定目录下生成app1的keytab文件。(xxx代表keytab文件存放路径可以自定义，kerberos.keytab文件名可以自己修改，app1代表创建的用户名)

CDH启用Kerberos

- **在CM中通过向导开启Kerberos认证**

① 登录cm点击Cluster1，启用Kerberos

- 注意配置Kerberos过程中，kdc管理员为xxx/admin@HADOOP.COM

Kerberos开发使用

- **Hive和Impala示例**

```
 */
public class ImpalaHiveTest {
    private static String connectionUrl;
    private static String jdbcDriverName;

    public enum Htype{
        HIVE("hive"),IMPALA("impala");
        public String name;
        Htype(String name){
            this.name = name;
        }
    }

        /**
         * 加载配置文件
         * @author admin
         * @throws IOException
         */
        private static void loadConfiguration() throws IOException {
                 Htype tag =Htype.IMPALA;
                 if(tag==Htype.IMPALA){
                       connectionUrl="jdbc:impala://node2:21051/dybdp_100;AuthMech=1;KrbRealm=DAYOU.COM;"
                            + "KrbHostFQDN=node2;KrbServiceName=impala;DelegationUID=dybdp";
                       jdbcDriverName = "com.cloudera.impala.jdbc41.Driver";
                 }else{
                       connectionUrl="jdbc:hive2://node2:10001/dybdp_200;principal=hive/node2@DAYOU.COM;"
                            + "hive.server2.proxy.user=hive";
                       jdbcDriverName="org.apache.hive.jdbc.HiveDriver";
                 }
        }

    public static <T> void main(String[] args) throws IOException {
                String sqlStatement = "create database test_1";
                loadConfiguration();
                System.out.println("\n==========================================");
                System.out.println("Cloudera Impala JDBC Example");
                System.out.println("Using Connection URL: " + connectionUrl);
                System.out.println("Running Query: " + sqlStatement);
        try {
```

Kerberos开发使用

- **Hive和Impala示例**

```
if (!new AtomicBoolean(false).getAndSet(true)) {
      if (System.getProperty("os.name").contains("Windows") || System.getProperty("os.name").contains("Mac"))
          System.setProperty("java.security.krb5.realm", "HADOOP.COM");
          System.setProperty("java.security.krb5.kdc", "192.168.1.203");}
      }
 org.apache.hadoop.conf.Configuration conf = new     org.apache.hadoop.conf.Configuration();
  conf.set("hadoop.security.authentication", "Kerberos");
 UserGroupInformation.setConfiguration(conf);
 UserGroupInformation.loginUserFromKeytab("hadoop@DAYOU.COM", "/Users/zhuwenjun/Downloads/hadoop.keytab");

 UserGroupInformation.getLoginUser().doAs(new PrivilegedAction<T>(){
     Connection con = null;
    @Override
    public T run() {
            try {
                Class.forName(jdbcDriverName);
                 con = DriverManager.getConnection(connectionUrl);
                Statement stmt = con.createStatement();
                stmt.execute(sqlStatement);
                ResultSet rs = stmt.executeQuery(sqlStatement);
                System.out.println("\n== Begin Query Results ======================");
                // print the results to the console
                while (rs.next()) {
                    // the example query returns one String column
                    System.out.println(rs.getString(1));
                }

                System.out.println("== End Query Results ======================\n\n");

                return null;
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }finally {
                try {
                    con.close();
                } catch (Exception e) {
                    // swallow
                }
```

Kerberos开发使用

- **Hive和Impala示例**
  - ../../Downloads/ImpalaHiveTest.java