Keytab Creation and Application Integration Guidebook

Table of Contents:

Introduction to Kerberos: A Trip to the Cinema

Imagine you want to see the latest action flick at your local stadium seating theater. Do you walk right in and go straight to a theater? Not quite. You stand in line at the box office, buy a ticket, and *then* go to the theater numbered on the ticket. You also don't save the ticket and try to use it the next day, since it's valid only for *that* theater on *that* day at *that* time.

Kerberos is like the box office between a client and something they want to access. If a client- like an application process or a user- wants to use something in a file system protected by Kerberos, the client must go to the Kerberos box office first, get a ticket, then present it to the file system.

In technical terms, Kerberos is a ticketing system used to secure computer resources. If you know what to do when you go to the movies, you also know how Kerberos works.

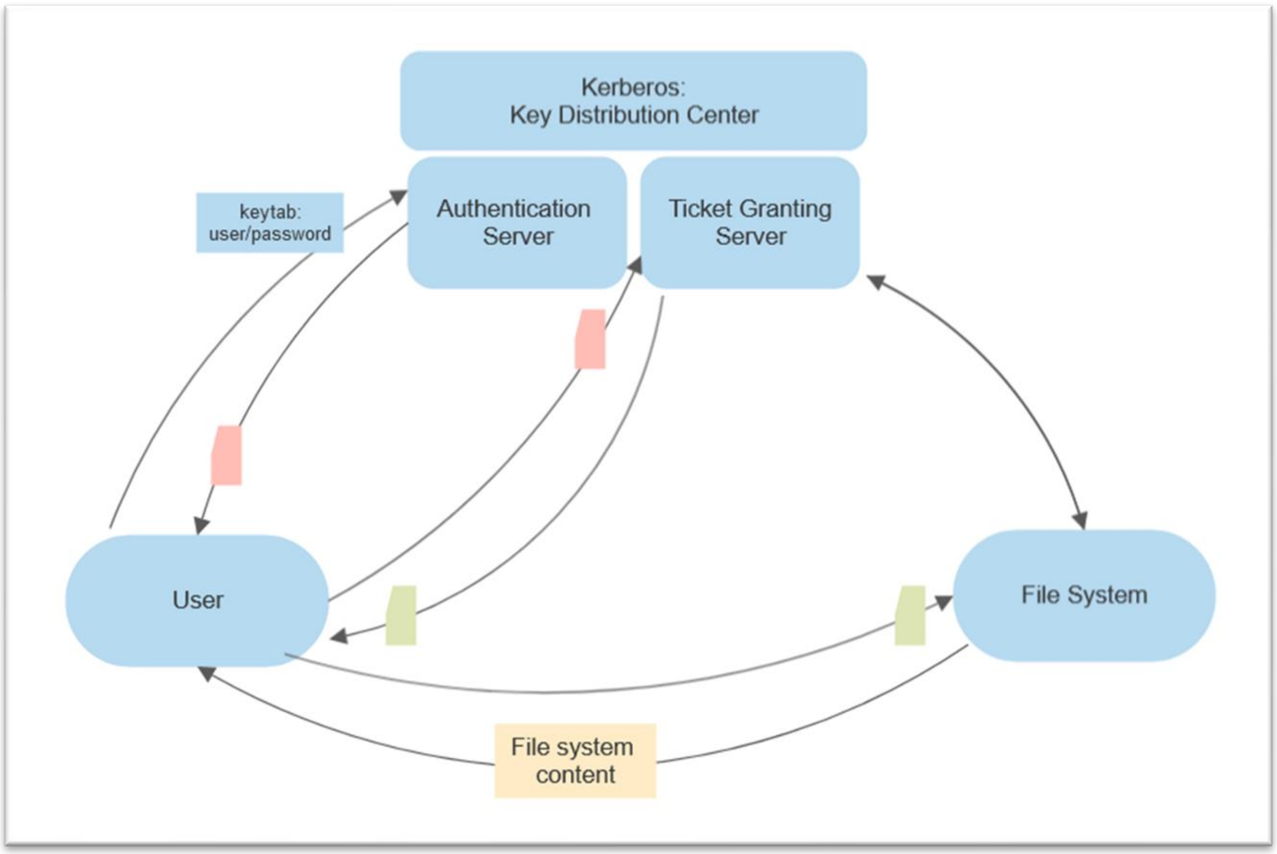
Kerberos does this in three sets of round-about trips from the client to a remote server and back again:

> **1: Authentication.** A client sends its username and password to the Kerberos server in an encrypted file called a **keytab**. Kerberos's authentication server decrypts it and checks if the username and password are valid. If they are, Kerberos sends a "ticket granting ticket" to the client. We'll call it Ticket #1.
>
> **2: Ticket Granting.** The client sends Ticket #1 back to Kerberos, but to a different part- to the ticket granting server. This accepts Ticket #1, which proves the client is who they say they are, and sends back Ticket #2, which allows the client to access a file system.
>
> **3. Authorization/Access.** The client sends Ticket #2 to the protected file system they want to access. The system accepts it and grants the client access to its resources. The ticket is valid for a limited time, typically one day.

Let's step through a high-level outline of the Kerberos protocol, then look at real-world implementation.

Kerberos, in Theory and Practice

First, the theory:

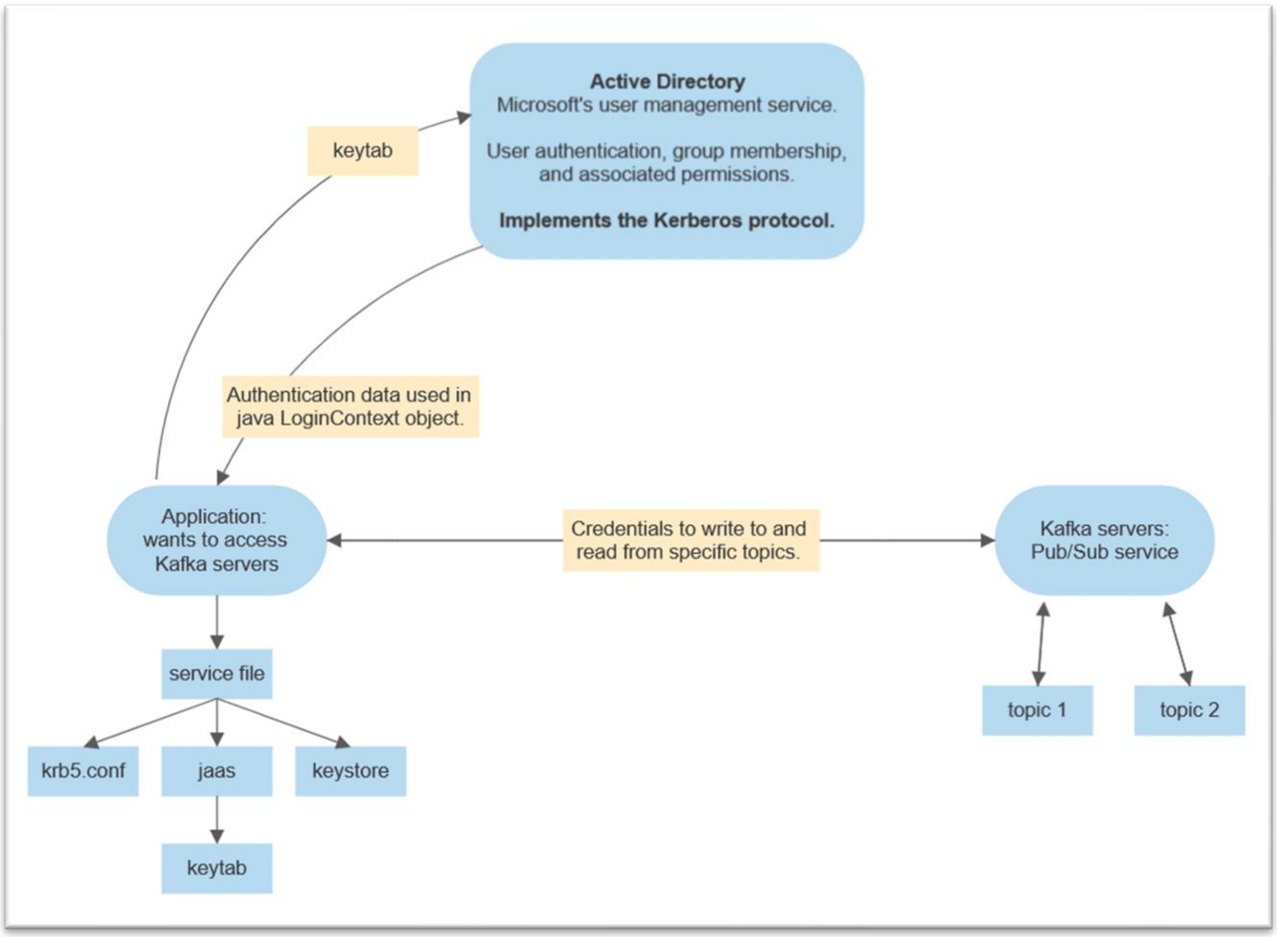


The Kerberos server is called the Key Distribution Center or KDC. You can think of “key” as another way of saying “ticket” in Kerberos-speak. Technically, an encryption key is passed along with the tickets, but we can leave that detail aside for now. Note that the KDC has two parts: the authentication server, which reads the keytabs, and the ticket granting server, which then gives clients tickets to access protected systems.

The first round of communication from User to KDC is like showing your ID to buy a ticket to an R rated movie at the box office. The second round is like handing that ticket to the usher, who tears it, gives you the stub, and points you to the right theater. The third round is like getting comfortable in your seat and enjoying the file system content.

The analogy is imperfect since Kerberos deals with two tickets, but the process is close enough. You go to one person who *authenticates* you, then go to another person who *authorizes* you, after which you can *access* the protected media.

Second, the practice:


Active Directory
Microsoft's user management service.
User authentication, group membership, and associated permissions.
Implements the Kerberos protocol.
keytab
Authentication data used in java LoginContext object.
Application: wants to access Kafka servers
Credentials to write to and read from specific topics.
Kafka servers: Pub/Sub service
topic 1
topic 2
service file
krb5.conf
jaas
keystore
keytab

This diagram seems to be missing pieces, but that's because Active Directory implements the Kerberos protocol behind the scenes. What's important is the Active Directory server acts as the box office, gatekeeping resources in XYZ INC's internal network to those who have valid tickets.

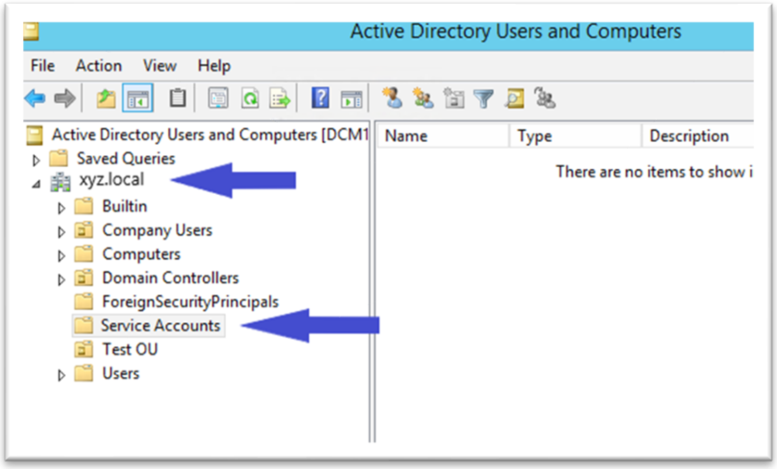
On the application side, configuration files need to be set up to tell the application where the keytab is on the local system and where the Active Directory server is on the network. The next section will go through that step by step.

On the protected file side, there are Apache Kafka servers. Kafka is a messaging queue. It stores data in plain-text log files it calls "topics". Apps can write data to a topic or read data from one. How all that works isn't important to Kerberos, though. What's important is a client can access the Kafka server's file system with a ticket granted by Active Directory (which implements the Kerberos protocol).

How to Create a Keytab

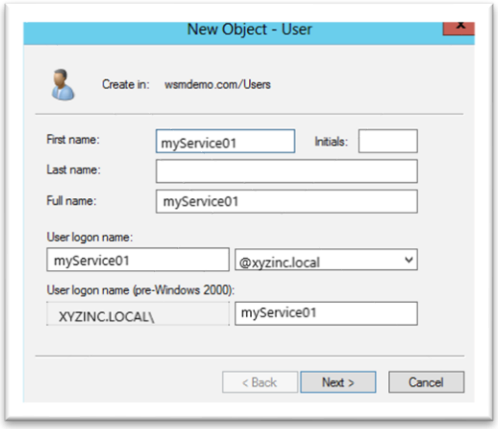
Create a service account in Active Directory

1. Log in to domaincontroller.xyzdev.local, the Active Directory domain controller.
2. On the desktop, double click **Active Directory Users and Computers**.
3. At the upper-right, click the triangle next to the xyz.local domain to expand it.
4. Right-click **Service Accounts**.



There are no items to show i

5. Click **New User**. The **New Object – User** window will open.



Create in: wsmdemo.com/Users

First name: myService01

Initials:

Last name:

Full name: myService01

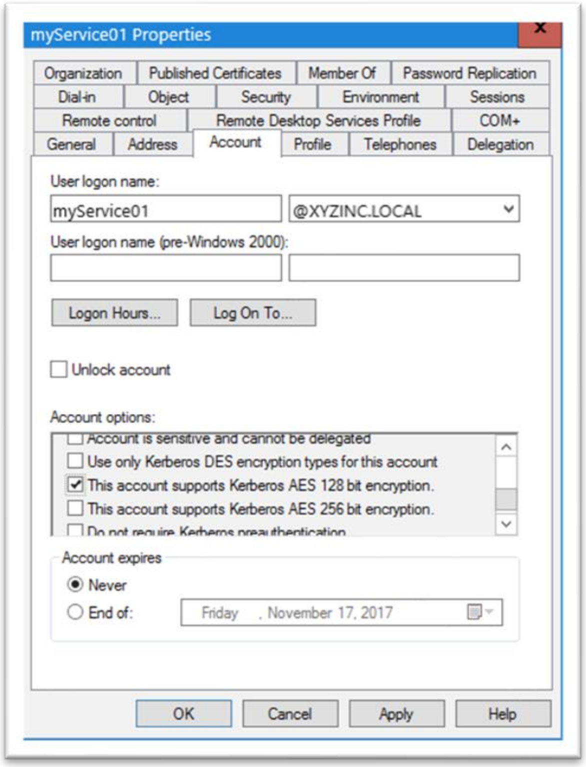
User logon name: myService01 @xyzinc.local

User logon name (pre-Windows 2000): XYZINC.LOCAL\ myService01

❖ **Info:** Though this entity is a User, an application will use it to authenticate with the Kerberos protocol. We call these types of AD Users “service accounts”.

6. Provide values for the fields below. Leave the rest blank or as-is.
   - **First Name**: myService01
   - **Full Name**: same as First Name.
   - **User Logon Name**: same as First Name.
7. Click **Next**.
8. Enter a password.
   ➢ **Note!** Store this password somewhere safe. You’ll need it to create the keytab.

9. Click the checkbox for **Password Never Expires**.
10. Ensure the other 3 checkboxes are unchecked.
11. Click **Next**.
12. Click **Finish**. The **New Object – User** window will close.
13. In the **Service Accounts** folder, right click the user you created and click **Properties**.
14. Click the **Account** tab.



User logon name:
myService01 @XYZINC.LOCAL
User logon name (pre-Windows 2000):

☐ Unlock account

Account options:
☐ Account is sensitive and cannot be delegated
☐ Use only Kerberos DES encryption types for this account
☑ This account supports Kerberos AES 128 bit encryption.
☐ This account supports Kerberos AES 256 bit encryption.
☐ Do not require Kerberos preauthentication

Account expires
● Never
○ End of: Friday , November 17, 2017

15. In the **Account Options** section, click the checkbox **This account supports Kerberos AES 128 bit encryption**.
16. Click **Apply**, then click **OK**.

Create a keytab file with that new service account

1. Log on to domaincontroller.xyzdev.local.
2. Open CMD as administrator.
3. Run the **ktpass** command below in the command prompt.
    - **Note!** A carrot preceded by a space (^) is how Windows signals a command continues on the next line.

```
ktpass ^
-princ <SPN> ^
-mapuser <AD-USER> ^
-pass <PASSWORD> ^
-crypto AES128-SHA1 ^
-ptype KRB5_NT_PRINCIPAL ^
-out C:\<KEYTAB-FILE-NAME>
```

ktpass is Microsoft's command line tool for making keytab files. It requires at least the following arguments:

- `-princ` is the Service Principal Name, or SPN. **This parameter is case sensitive.**
- `-mapuser` is the User Logon Name of the Active Directory account.
- `-pass` is the password of the Active Directory account.
- `-crypto` defines what kind of key is created in the keytab.
- `-ptype` defines the principal type. We use the general type here.
    - **Info:** A "principal" is something Kerberos can assign tickets to, uniquely identified by its SPN. In this case, the principal is the service account we're making a keytab for.
- `-out` defines the output file path and name of the keytab file ktpass creates.

How to Set Up a Keytab for Application Use

A few things need to be set up before an application can use a keytab. Below is a list of what's involved. The next section will walk you through creating and using these files.

- **file.keytab: The keytab file.** This is where a username and password are encrypted. It's like an ID card for the user or application.

- **krb5.conf: the Kerberos (version 5) configuration file.** This defines where the Kerberos server is on the network. In our case, it points to an Active Directory server.

- **keystore.jks: the keystore (also called the trust store).** This stores the encryption keys that are passed between the client and the Kerberos server.

- **myApp_jaas.conf: the jaas file (not to be confused with "jazz" file).** The *Java Authentication and Authorization Service* file. It's what java applications use to log in to something. It contains the file path to the keytab and the SPN of its principal.
  - ❖ Recall that a **principal** is something Kerberos grants tickets to. In this case it's the service account user, whose name and password are embedded in the keytab.
  - ❖ Recall that the **SPN** is how Kerberos uniquely identifies a principal.

- **myapp.service, the Linux systemd service file.** This is how an application is set to run as a service with systemctl. It also defines actions and environment variables on service startup. This is what points an application to the jaas file and the keystore.
  - ❖ **systemd** is a set of service features most linux OS's use.
  - ❖ **systemctl** is a command line utility used to start, stop, and check up on running processes. It uses .service files to determine what happens when a service is started or stopped.

System setup: krb5.conf and keystore.jks

These two files enable a client to communicate with the Kerberos server:

- krb5.conf defines where and how to find the Kerberos server on the network.
- keystore.jks stores the keys the client and Kerberos pass to each other.

These are system level config files. It's likely they'll already be set up on the machine you're working on, either as part of a VM golden image, a base docker container, or an ansible configuration script.

If you need to set these up, though, it's easy to do. The krb5.conf is the same on all machines that need to talk to a particular Kerberos server. You can copy it from another machine that has it in /etc/krb5.conf.

You can also copy the keystore.jks file from /etc/security/keystore.jks on a machine that has it if you're able to access the security folder. As an Operations engineer you should have sudo access on all virtual machines. If you do not, talk to your manager.

krb5.conf template

```
[logging]
default = FILE:/var/log/krb5libs.log

[libdefaults]
dns_lookup_realm = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
rdns = false
default_ccache_name = KEYRING:persistent:%{uid}
default_tkt_enctypes = aes128-cts-hmac-sha1-96
default_tgs_enctypes = aes128-cts-hmac-sha1-96

default_realm = DOMAIN.LOCAL

[realms]
DOMAIN.LOCAL = {
      kdc = xyzincdc.domain.local
      admin_server = xyzincdc.domain.local
      default_domain = DOMAIN.LOCAL
}

[domain_realm]
.domain.local = DOMAIN.LOCAL
domain.local = DOMAIN.LOCAL
```

Application Setup: jaas file, service file, and java app code

There are two things Operations engineers need to set up for an application to use a keytab. Each app will have its own instance of each:

- A jaas file on every machine the app is installed on.
- A service file on every machine the app is installed on (which points to the jaas file).

The list below contains examples of the values you'll need to know to set up these files. Try to find out what they are for a new app before you get started. Most often you'll only need to ask the developers what they've named their application .jar file. The other values are templates the name of the .jar plugs into or are standard file paths.

Values used in these examples:

krb5.conf file path	`/etc/krb5.conf`
keystore file path	`/etc/secuirty/keystore.jks`
jaas file path	`/etc/security/myApp_jaas.conf`
SPN	`myApp/xyzinc.local@XYZINC.LOCAL`
keytab file path	`/etc/security/myApp.keytab`
.jar file path	`/opt/myApp/myApp.jar`

Finding the SPN of a keytab file

You might need to know an SPN when troubleshooting issues or if a developer isn't available to tell you what it should be for an application. The **klist** utility can reveal it to you.

1. Run `cd /path/to/myApp.keytab`
2. Run `klist -k -t myApp.keytab`
   - If this command isn't found, run:
     `sudo yum install krb5-workstation -y`

   `-k` means to list the keys encrypted in the keytab.

   `-t` means to display time stamps related to each key in the keytab.

The SPN is the value of the **Service principal** field on the bottom-right:

```
Valid starting     Expires            Service principal
01/01/23 19:49:21  06/08/23 05:49:19  myServiceAccount/xyzinc.local@XYZINC.LOCAL
```

Create a jaas file for a .jar application named myApp

1. Copy the jaas file template below.
2. Paste it in Notepad++ or any text editor that removes character metadata.
3. Update the following values:
   a. `keyTab` = the absolute file path to the keytab.
   b. `principal` = the SPN of the keytab.
4. Copy your updated jaas file text.
5. Log in to a machine and run `sudo su` to become root.
6. Run `cd /etc/security`
7. Run vim `serviceName_appName_jaas.conf`
8. Type `i` to enter interactive mode.
9. Paste in the jaas file text.
10. Type `Shift + :` to enter vim's command prompt.
11. Type `wq` to write and quit.

jaas file template

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
renewTicket=true
debug=true
storeKey=true
useKeyTab=true
keyTab="/etc/security/myApp.keytab "
principal="<SPN>"
};
```

The next step is to create a service file. This does two things:

- Ensures the application starts when the OS is started (even after a reboot).
- Defines variables the application will use (like file paths to important files).

Create a Linux service file for a .jar app named myApp

1. Copy the service file template below.
2. Paste it in Notepad++ or any text editor that removes character metadata.
3. Update the following values:
   a. `ExecStart` = the path to the .jar on the file system.
   b. `security.auth.login.config` = the path to the jaas.conf file.
4. Copy the text of your updated service file.
5. Log in to a machine and run `sudo su` to become root.
6. Run `cd /etc/systemd/system`
7. Run `vim mypp.service`
8. Type `i` to enter interactive mode.
9. Paste in your updated service file text.
10. Type `Shift + :` to enter vim's command prompt.
11. Type `wq` to write and quit.
12. Run `systemctl enable myApp.service` to make this service start on system startup.
    a. **Note!** If you need to edit a service file later on, you must run `systemctl daemon-reload` for your changes to take effect.

Linux service file template

```
[Unit]
Description=myApp
After=syslog.target

[Service]
ExecStart=/opt/myApp/myApp.jar
SuccessExitStatus=143
Environment="JAVA_OPTS=-
Djava.security.auth.login.config=/etc/security/myApp_jaas.conf -
Djava.security.krb5.conf=/etc/krb5.conf -
Dkakfa.producer.security.truststore.location=/etc/security/keystore.jk
s"

[Install]
WantedBy=multi-user.target
```

And you're done!

That's everything you need to set up from an Operations perspective. Developers are responsible for adding code to their java applications that locates the jaas file, imports it, and uses the information in it to create a LoginContext object. Developers are free to do that as they wish, working within the standards set above on where the keytab and jaas files are on the application servers.