# KNIME Server Advanced Setup Guide

KNIME AG, Zurich, Switzerland

Version 4.18 (last updated on )

# Table of Contents

# Introduction

This guide covers advanced topics of a KNIME Server deployment, setup and configuration in an enterprise environment.

If you are looking to install KNIME Server, you should first consult the KNIME Server Installation Guide.

For guides on connecting to KNIME Server from KNIME Analytics Platform, or using KNIME WebPortal please refer to the following guides:

- KNIME Server User Guide

- KNIME WebPortal User Guide

For all regular administration configuration options and a basic understanding of KNIME server please consult the KNIME Server Administration Guide.

In the following it is assumed that you have a knowledge of all topics covered in the previously mentioned guides.

# Enterprise User Authentication

User authentication in an enterprise environment is usually done through some centralized service. The most used service is LDAP. LDAP authentication is the recommended authentication in any case where an LDAP server is available. If you are familiar with your LDAP configuration you can add the details during installation time, or edit the `server.xml` file post installation. If you are unfamiliar with your LDAP settings, you may need to contact your LDAP administrator, or use the configuration details for any other Tomcat based system in your organization. This section describes how to set up KNIME Server for LDAP authentication.

Another possibility of user authentication is single-sign-on. KNIME Server can be configured to support Kerberos authentication in combination with LDAP. This section also contains steps for a simple Kerberos setup.

## Configuring an LDAP connection for KNIME Server

KNIME Server manages all user authentication by the built-in mechanisms of Apache Tomcat. Therefore the most comprehensive documentation for configuring authentication is the Apache Tomcat Realm Configuration HOW-TO. Specifically for information about LDAP (also Active Directory) configuration, see section JNDIRealm.

**Terminology**

> Throughout this document we refer to establishing an *LDAP connection*, *LDAP account* etc. Since one of the popular ways to manage user authentication is Microsoft Active Directory, which supports LDAP, you may want to substitute *LDAP account* for *Active Directory account*.

### Quickstart

In most cases it should be possible to contact your local LDAP/Active Directory administrator; they should be able to provide the necessary information.

You can ask for the following:

1. Do they already have configuration details for a Tomcat server? If so, this connection information can be reused.

2. LDAP Connection information (Hostname, Port, is TLS/SSL used?).

3. Whether they are using bind mode, or comparison mode.

---

4. How the group information is stored.

They will need to provide configuration that can fit into a template like this:

```
<Realm   className="org.apache.catalina.realm.JNDIRealm"
      connectionURL="ldap://localhost:389"
        userPattern="uid={0},ou=people,dc=mycompany,dc=com"
          roleBase="ou=groups,dc=mycompany,dc=com"
          roleName="cn"
        roleSearch="(uniqueMember={0})"
/>
```

This information is added to the `server.xml` file which is found in `<apache-tomcat>/conf/server.xml`.

A restart of the Apache Tomcat process and KNIME Server is required for the changes to the configuration file to take effect.

## Advanced Troubleshooting

The remaining sections of this documentation describe how to setup an LDAP connection for KNIME Server. This is only intended as a way to gather related information into one place, not as comprehensive documentation for either LDAP or Tomcat.

The first prerequisite is Apache Directory Studio, or some other LDAP configuration tool. We use Apache Directory Studio to do the testing. The benefit of using this tool is that it is open source, free to download, works on Windows/Linux/Mac, so a customer can download the software and do queries to get started.

We will follow three basic steps:

1. LDAP Connection information (Hostname, Port, SSL?).

2. Whether they are using bind mode, or comparison mode.

3. How the group information is stored.

### LDAP Connection information (Hostname, Port, SSL)

To establish a connection to an LDAP server you'll need to know:

- The LDAP server hostname (or IP)

- Whether the server uses SSL secured connections or not

- Which port is being used — default ports are 389 for LDAP (unencrypted, or encrypted by TLS) and 636 for LDAPS (SSL secured)

## Setup Apache Directory Studio to browse your LDAP directory

### Setup connection to server
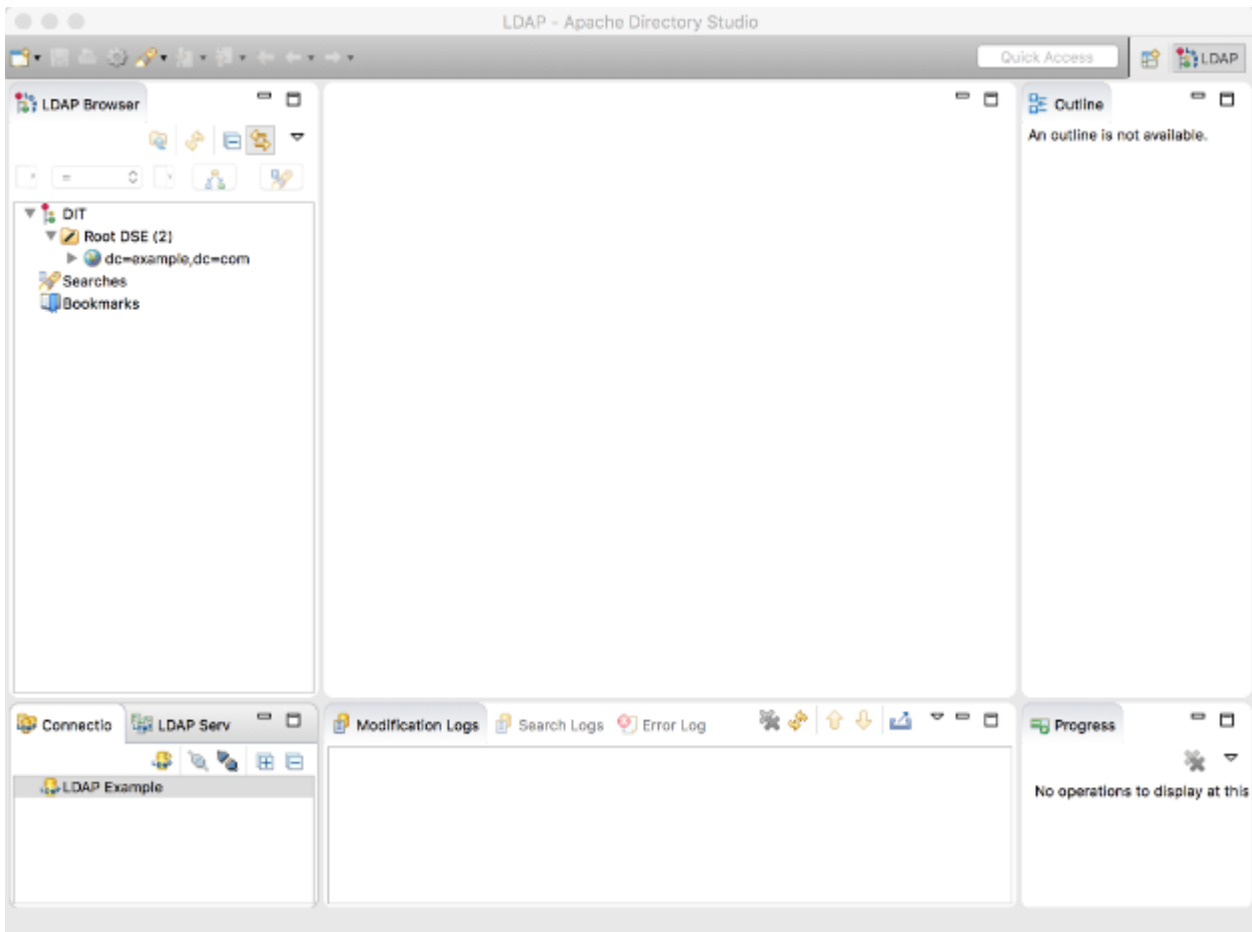


Add in the connection details of your LDAP server

Setup connection to LDAP server

Note that we don't use authentication here. Typically, you will need to authenticate, and in most cases this can be your LDAP username and password.

Setup connection

You can click 'Fetch Base DNs' to autopopulate the answers. In our example the Base DN is `dc=example,dc=com`. This will vary, for example knime.com might use the Base DN `dc=knime,dc=com`.
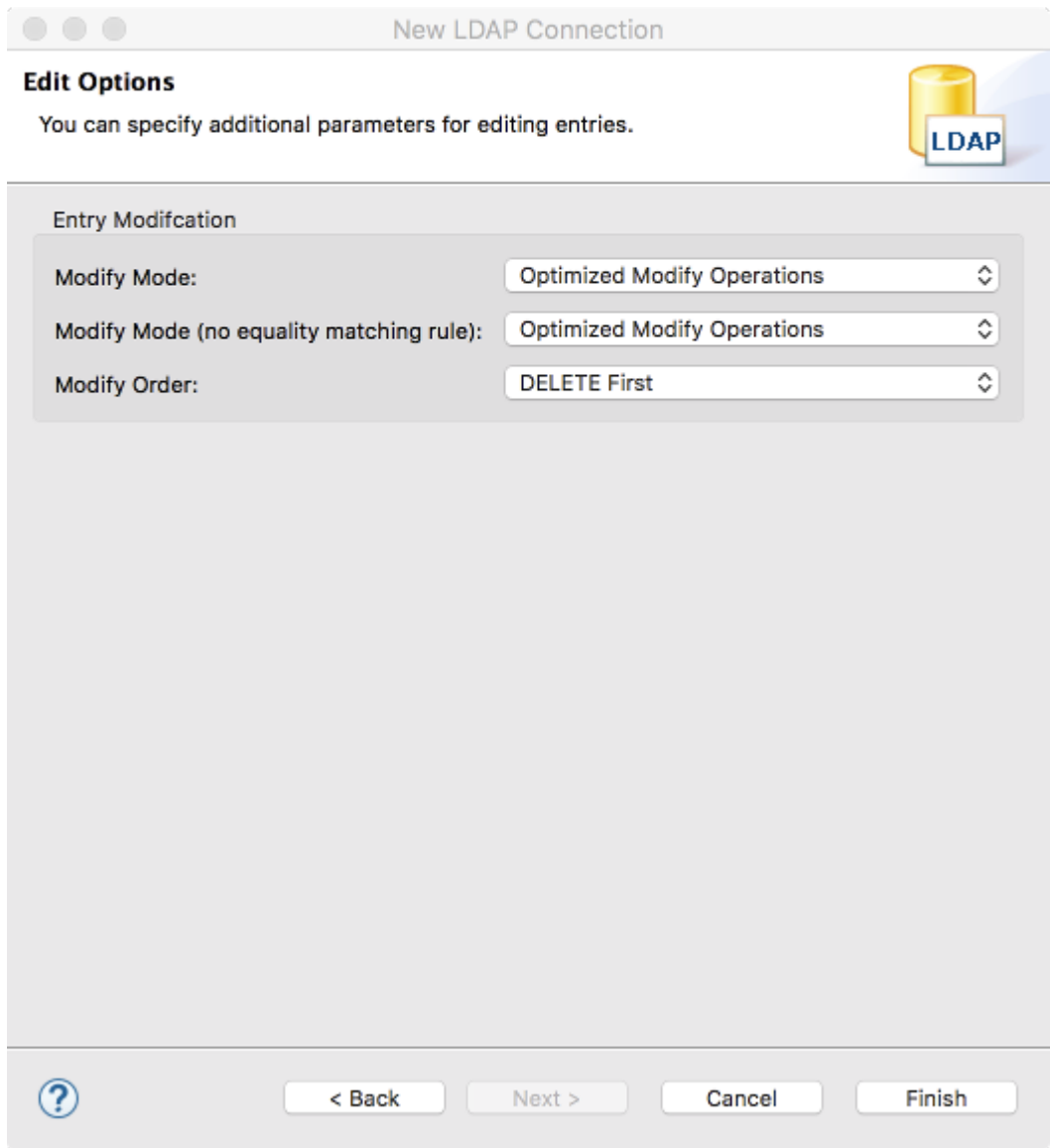
Finalize connection

You can leave the next page as is, and click Finish.

## Browse LDAP Tree

The LDAP Browser is now populated, and you can begin browsing the LDAP directory.

Determine information required for KNIME/Tomcat LDAP configuration

First refer to the Tomcat documentation on LDAP. The documentation is very comprehensive, we distilled some of the key points below. For full details refer to the Tomcat documentation.

Basically we need to construct something that looks like:

```
<Realm    className="org.apache.catalina.realm.JNDIRealm"
       connectionURL="ldap://52.50.222.127:389"
         userPattern= TOBEDETERMINED
            roleBase= TOBEDETERMINED
            roleName= TOBEDETERMINED
          roleSearch= TOBEDETERMINED
/>
```

We already know the `connectionURL`, since this was required to setup Apache Directory Studio.

Next we need to determine the `userBase` property. The first item in the tree is usually the Base DN, which will define the `userBase` property.
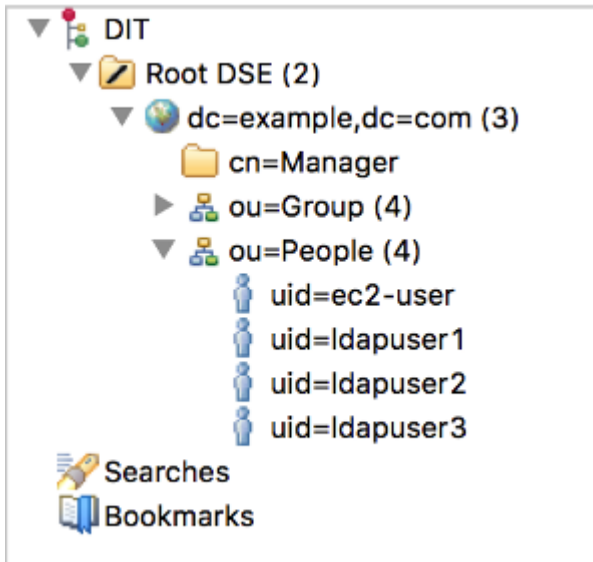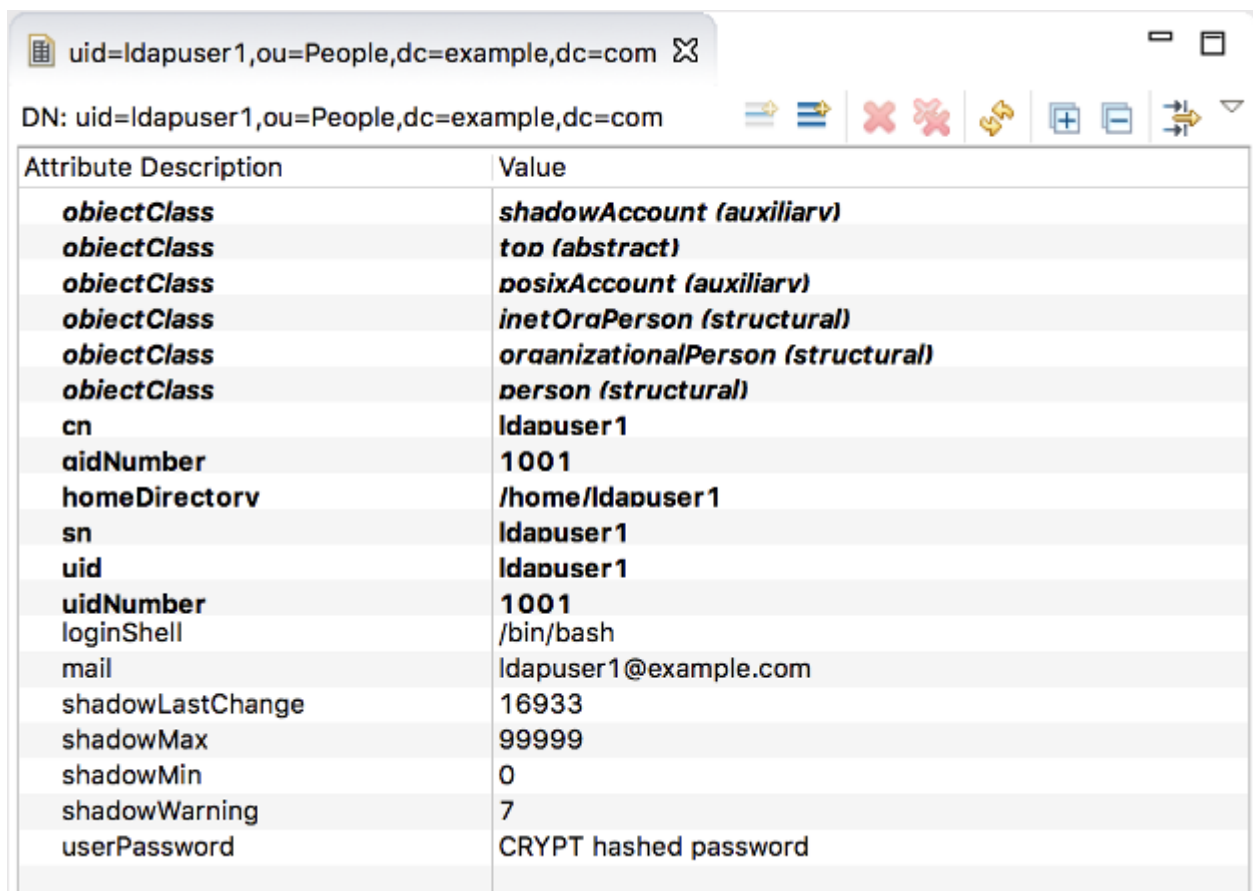


You can browse the tree to find the users. In our case `ou=People`. Expanding the subtree shows the list of users. In our case there are four users (*ec2-user*, *ldapuser1*, *ldapuser2*, *ldapuser3*).

## Determine whether users are checked by bind mode, or comparison mode

### Bind Mode

In our case, if users log in as e.g. *ldapuser1* (the username is the same as the key).

We already know the Base DN, and looking at the user information we see that the uid is the username that we want to use to authenticate. So we can construct the `userPattern`.

Use the `userPattern`: `uid={0},ou=people,dc=example,dc=com`

So the example would look like:

```
<Realm    className="org.apache.catalina.realm.JNDIRealm"
      connectionURL="ldap://52.50.222.127:389"
        userPattern="uid={0},ou=people,dc=example,dc=com"
          roleBase=TOBEDETERMINED
          roleName=TOBEDETERMINED
        roleSearch=TOBEDETERMINED
 />
```

Note that we still don't know how to specify `roleBase`, `roleName`, `roleSearch`. We'll come back to that later.

## Comparison Mode

In this case there is no one-to-one mapping between the login name and the username, we want to use e.g. the email address category. In this example that is `ldapuser1@example.com`.

To perform this kind of login, we need comparison mode:

Here the Base DN is needed for `userBase`, and we also need to define `userSearch`. Here we are searching for `mail`.

```
<Realm   className="org.apache.catalina.realm.JNDIRealm"
     connectionName="cn=Manager,dc=example,dc=com"
connectionPassword="secret"
     connectionURL="ldap://52.50.222.127:389"
        userBase="ou=people,dc=example,dc=com"
      userSearch="(mail={0})"
    userRoleName="memberOf"
        roleBase= TOBEDETERMINED
        roleName= TOBEDETERMINED
      roleSearch= TOBEDETERMINED
/>
```

## Group access

Now that users are authenticated, we need to configure the groups that have access:

For that we will need the `roleBase` and the `roleName` parameters. You can browse the `ou=Group` tree for more information. Here let's take the example that the `hrpeople` group

should be able to access KNIME Server.
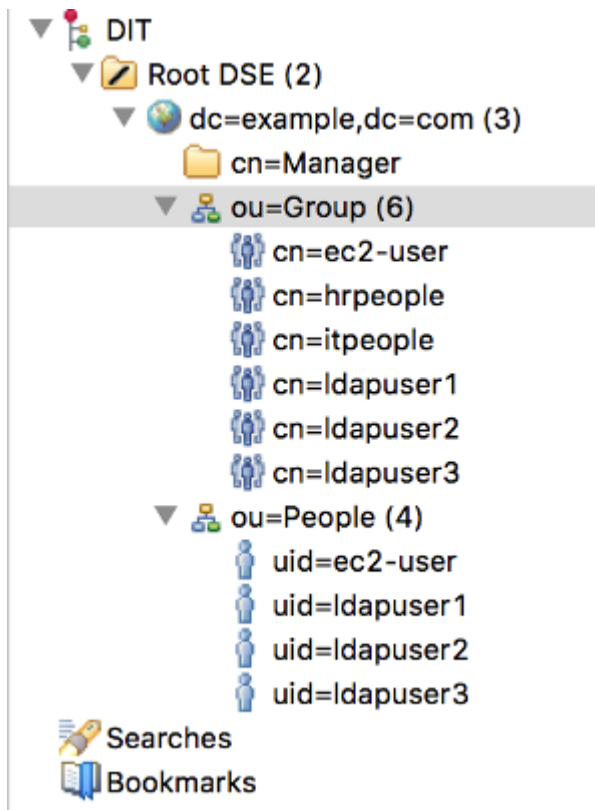




In the example, value is member that we want to search for is 'member'.

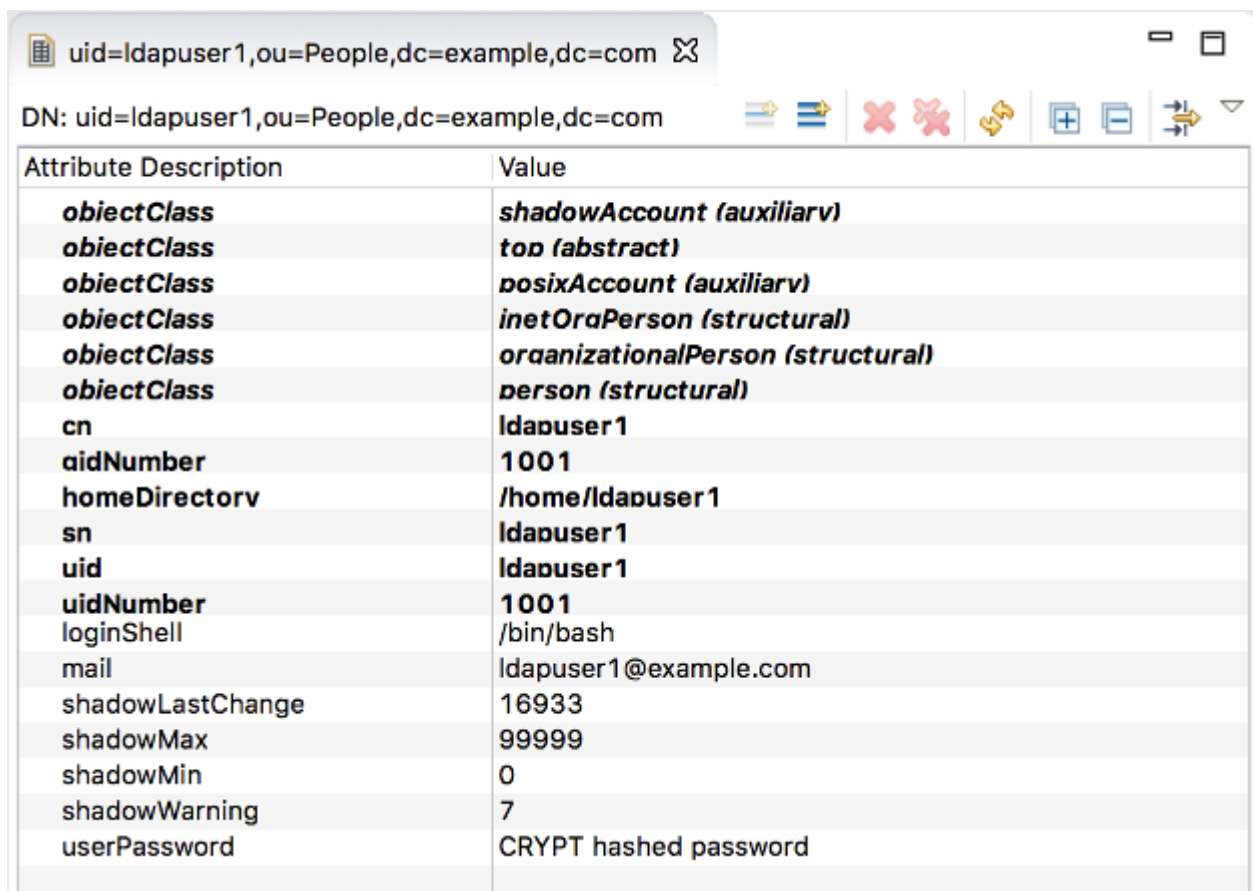Which leads to the configuration:

```
<Realm    className="org.apache.catalina.realm.JNDIRealm"
        connectionURL="ldap://52.50.222.127:389"
            userBase="ou=people,dc=example,dc=com"
          userSearch="(mail={0})"
        userRoleName="memberOf"
            roleBase="ou=Group,dc=example,dc=com"
            roleName="cn"
          roleSearch="(member={0})"
  />
```

There is a second possibility where group membership is stored in the user data (this is uncommon, and not covered in this guide. See the full Tomcat documentation).

Nested roles (where a role/group can contain other roles/groups) are also possible, in which case add the `roleNested` parameter. E.g. Group 'IT', contains some usernames, plus 'Windows', 'UNIX', 'Mac' groups. Those groups may also contain sub-groups.

Hopefully you now have the details that you need to connect KNIME Server to LDAP.

## Active Directory Example

If you are using Active Directory as your user database and sticked to the default structure, the following configuration serves as a good starting point:

```
<Realm className="org.apache.catalina.realm.JNDIRealm"
    connectionName="cn=Manager,dc=example,dc=com"
    connectionPassword="secret"
    connectionURL="ldap://52.50.222.127:389"
    userSubtree="true"
    userBase="cn=Users,dc=domain,dc=com"
    userSearch="(sAMAccountName={0})"
    userRoleName="memberOf"
    roleBase="cn=Users,dc=domain,dc=com"
    roleName="cn"
    roleSearch="(member={0})"
    roleSubtree="true"
    roleNested="true"/>
```

You have to adjust the three highlighted connection parameters, as well as the two `dc` values in the `userBase` and `roleBase`. The other parameters can usually be used as they are.

## Combined Realm

It is possible to setup a combined realm where both the user database and LDAP authentication are used in parallel. Generally this is not recommended, but can be useful for debugging and initial setup/testing. The example below shows how this might work.

```
<Realm className="org.apache.catalina.realm.LockOutRealm">
    <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
          resourceName="UserDatabase"/>
    <Realm   className="org.apache.catalina.realm.JNDIRealm"
        connectionURL="ldap://52.50.222.127:389"
            userBase="ou=people,dc=example,dc=com"
          userSearch="(mail={0})"
        userRoleName="memberOf"
            roleBase="ou=Group,dc=example,dc=com"
            roleName="cn"
          roleSearch="(member={0})"/>
</Realm>
```

## Encrypted LDAP

In case you are using encrypted LDAP authentication and your LDAP server is using a self-signed certificate, Tomcat will refuse it. In this case you need to add the LDAP server's certificate to the global Java keystore, which is located in `<jre-directory>/lib/security/cacerts`:

```
keytool -import -v -noprompt -trustcacerts -file
<server certificate> -keystore <jre>/lib/security/cacerts -storepass changeit
```

Alternatively, you can copy the `cacerts` file, add your server certificate, and add the following two system properties to `<apache-tomcat>/conf/catalina.properties`:

```
javax.net.ssl.trustStore=<copied keystore>
javax.net.ssl.keyStorePassword=changeit
```

## Troubleshooting

In some cases you will want to extract additional log file information about the LDAP authentication process. In this case you can edit `apache-tomcat*/conf/logging.properties` to add:

```
org.apache.catalina.realm.level = ALL
org.apache.catalina.realm.useParentHandlers = true
org.apache.catalina.authenticator.level = ALL
org.apache.catalina.authenticator.useParentHandlers = true
```

Once you have made the changes you will need to restart the knime-server process/service.

When you have successfully debugged your problem, don't forget to comment out or remove these lines from the `logging.properties` file, as it will create unnecessarily large log files.

# Configuring Single-Sign-On with Kerberos and LDAP

Single-Sign-On can be configured for KNIME Server. This includes the WebPortal, but also all other services (REST, SOAP, etc.) KNIME Server provides.

The technology used to achieve this is Kerberos, which is a network protocol used for authentication by the means of tickets and strong encryption. In the following it is assumed that you are familiar with the basic concepts of Kerberos and LDAP, as explained in the section before. You can find comprehensive documentation for the latest version of Kerberos here.

This section describes step-by-step how to set up Kerberos authentication by the means of an **Active Directory** service and **Windows clients**. Other setups are possible and may require different procedures to be functional.

> **i** Most setups will deviate in certain aspects from this guide, so make adjustments where necessary.

Kerberos requires setup for all three parties involved: the Kerberos and LDAP service (Active Directory), the Tomcat server running KNIME Server, and the clients.

## Active Directory Configuration

The first step is to set up the Active Directory correctly. It is assumed that you already have an Active Directory domain with users and correct groups for KNIME Server usage set up. Additional steps specific to Kerberos are:

1. Create a technical user for the Tomcat server in LDAP.

2. Associate a Service Principal Name (SPN) on with the newly created user for the Tomcat server. To do so, open a Windows PowerShell and enter:

```
setspn -s HTTP/TOMCAT_FQDN@REALM TECHNICAL_USER
```

In the above command, replace

- `TOMCAT_FQDN` with the fully qualified domain name (FQDN) of the machine that runs KNIME Server (and thus the Tomcat server),

- ◦ `REALM` with the Kerberos realm of your Active Directory installation,

- ◦ and `TECHNICAL_USER` with the name of the technical user you have created in the previous step.

  It is important that for the `TOMCAT_FQDN` the DNS entries (FQDN to IP) as well as reverse DNS entries (IP to FQDN) entries can resolved by the domain controller **and** all clients.

3. Make sure that the right encryption methods are active on the domain controller:
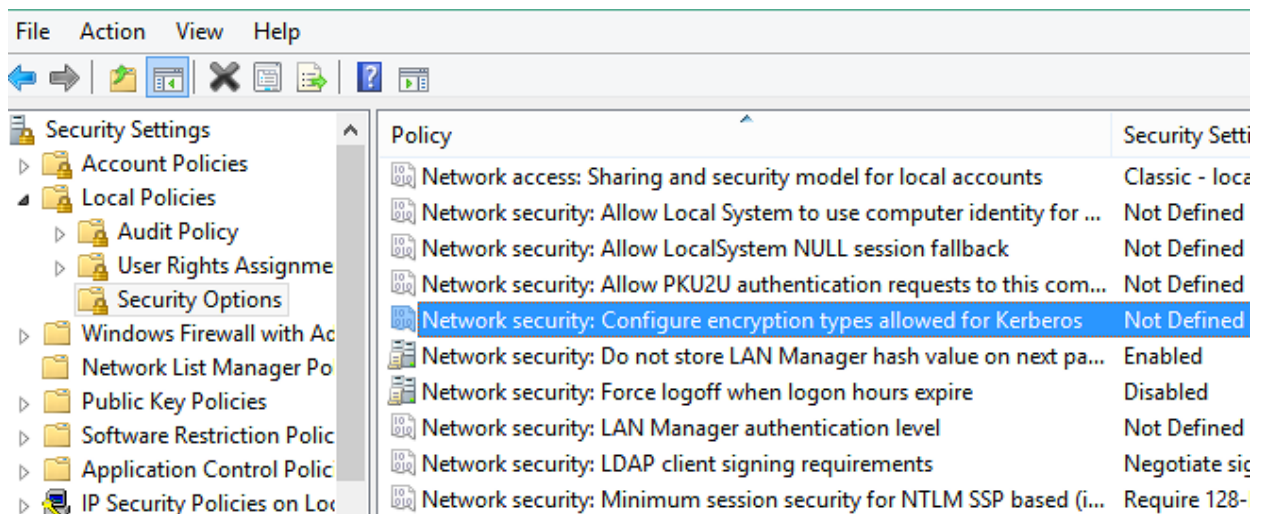


a. Go to *Administrative Tools → Local Security Policy*

b. Browse to *Security Settings/Local Policies/Security Options*

c. Find the entry *Network security: Configure encryption types allowed for Kerberos*. If the value is not defined, then all encryption types are allowed. If it is defined, make sure it contains at least the methods: `RC4_HMAC`, `AES128`, `AES256` and `Future Encryption Types`.

4. Open a Windows PowerShell and create a `keytab` file using the following command. Adjust the values according to your settings:

```
ktpass /out PATH/tomcat.keytab
  /mapuser TECHNICAL_USER@REALM
  /princ HTTP/TOMCAT_FQDN@REALM
  /Pass +rndPass
  /crypto AES256-SHA1 ptype KRB5_NT_PRINCIPAL
```

The created keytab file needs to be copied to the Tomcat server later.

5. Open the "User Properties" in Active Directory for the technical Tomcat user you have created. Then go to the "Account" tab and make sure the following settings are set:
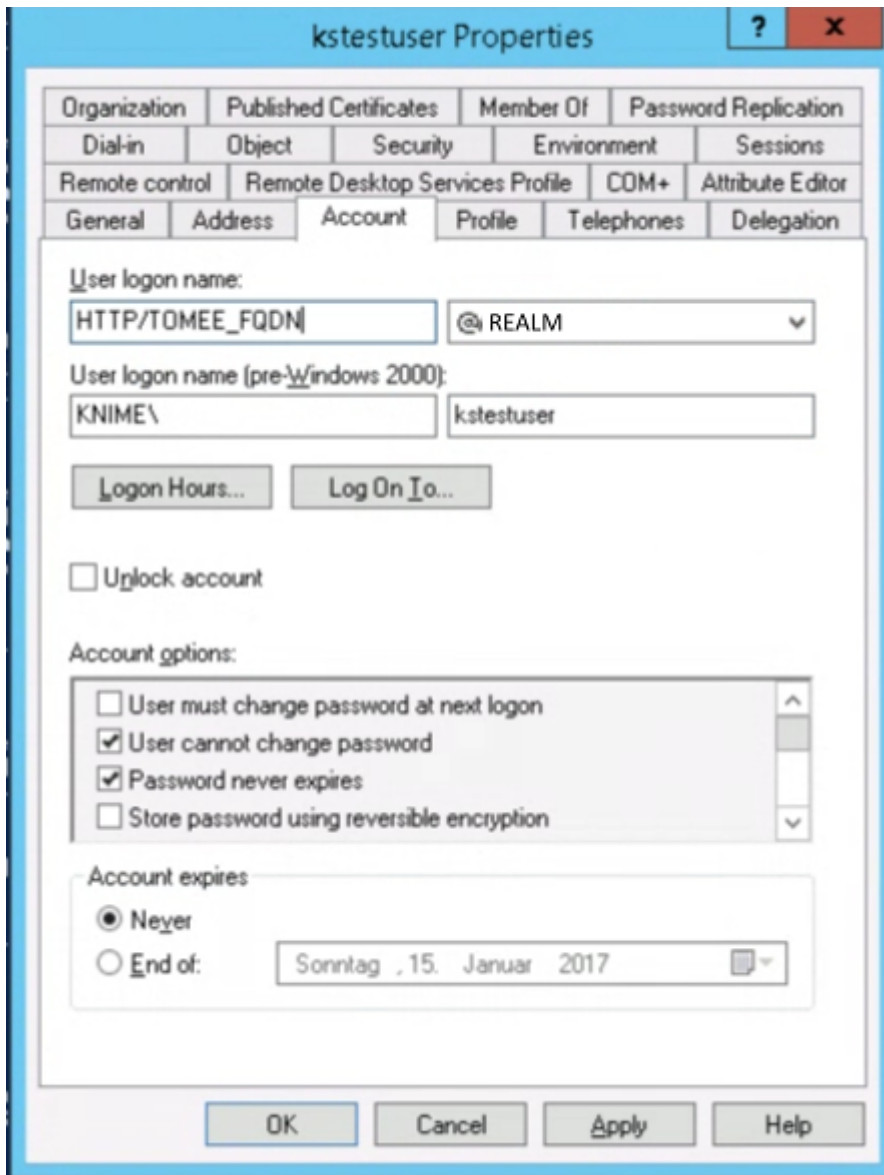
a. User logon name is correctly set

b. Password never expires = `true`

c. User cannot change password = `true`

d. This account supports Kerberos AES 128 bit encryption = `true`

e. This account supports Kerberos AES 256 bit encryption = `true`

f. Use Kerberos DES encryption for this account = `false` (recommended)

6. Then go to the "Delegation" tab and set the radio button to *Trust this user for delegation to any service (Kerberos only)*

## Tomcat Server Configuration

1. Install KNIME Server as outlined in the KNIME Server Installation Guide.

---

2. Make appropriate configuration adjustments as explained in the KNIME Server Administration Guide.

3. Setup LDAP authentication in the `server.xml` to connect to your Active Directory, as described in Configuring an LDAP connection for KNIME Server . Note that it might be necessary to create a temporary listing user to perform the LDAP lookups. This step is optional, but recommended to test that basic LDAP authentication is functional.

4. Verify that the environment variables `JAVA_HOME` and `CATALINA_HOME` are properly defined:

   ◦ `JAVA_HOME` should point to the JDK home directory (containing a `bin` folder)

   ◦ `CATALINA_HOME` should point to the Tomcat directory (containing a `bin` folder).

   On Windows this can be done in *Control Panel → System → Advanced system settings*
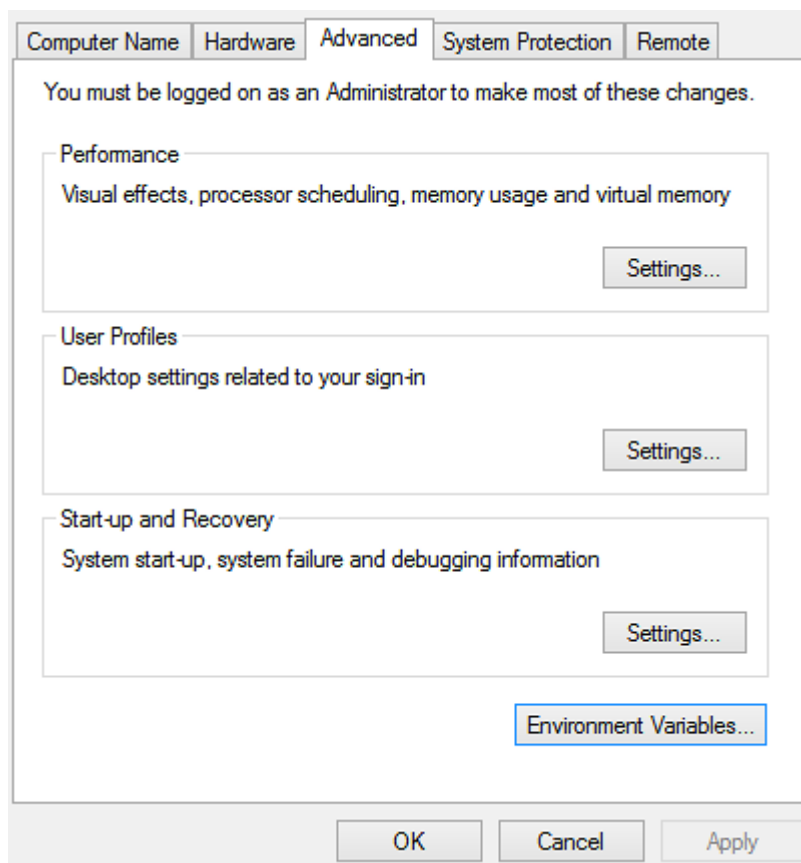


   a. Click on Environment Variables

   b. In the System Variables group check for the existence of `JAVA_HOME` and `CATALINA_HOME`. Create or adjust the values accordingly.

   On Linux create or change the values in `/etc/default/knime-server`

5. Once a working standard LDAP setup has been verified, make a backup of the contents

of `<CATALINA_HOME>/conf` by copying it to `<CATALINA_HOME>/conf_ldap`.

6. On Windows, open *regedit* and do the following:

   a. Navigate to
   `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Kerberos\Paramet`
   `ers`.

   b. Add the key `allowtgtsessionkey` (`REG_DWORD`) and set the value to 1.
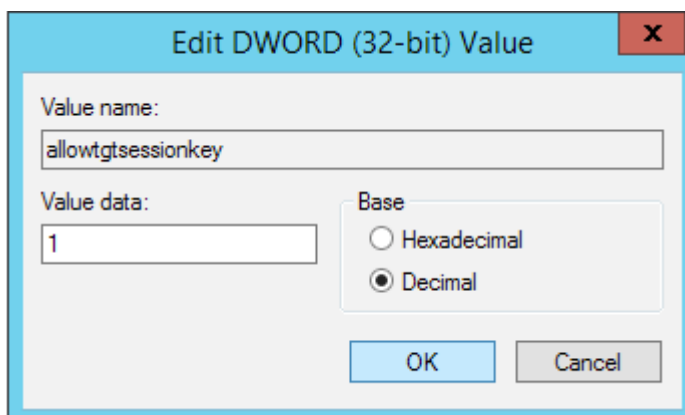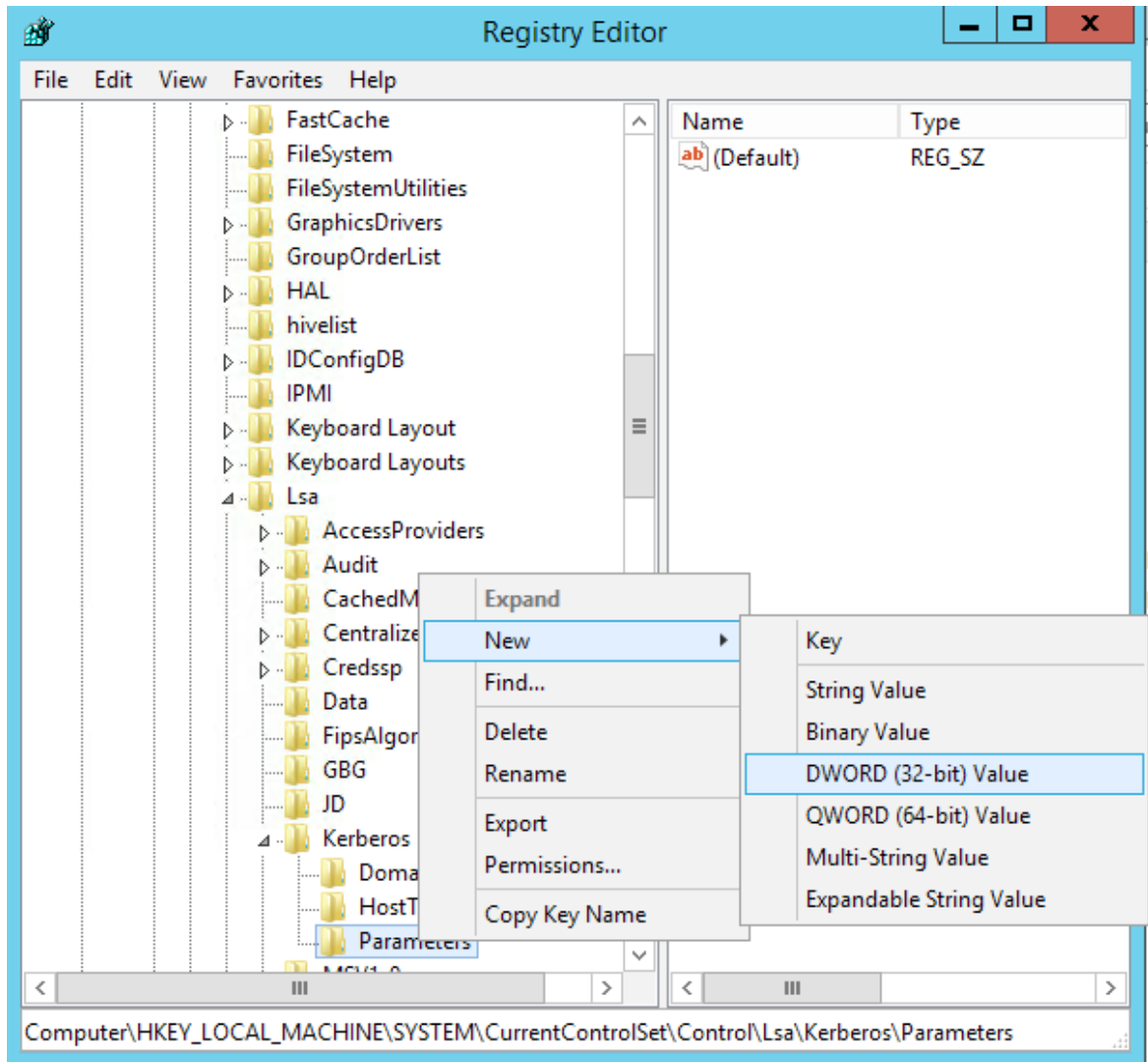
7. On Windows, make sure that the right encryption methods for Kerberos are active:

   a. Go to *Administrative Tools → Local Security Policy*

   b. Browse to *Security Settings/Local Policies/Security Options*

   c. Find the entry *Network security: Configure encryption types allowed for Kerberos*
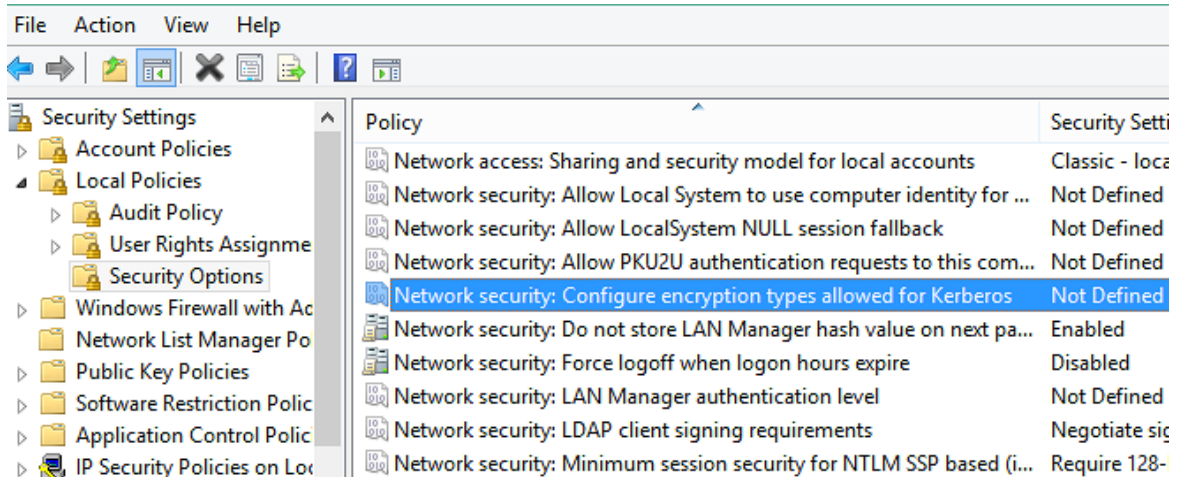


   d. If the value is not defined all encryption types are allowed. If it is defined, make sure it contains at least the methods: `RC4_HMAC`, `AES128`, `AES256` and `Future Encryption Types`.

8. Copy the previously created keytab file for the SPN to a location of your choosing. Recommended would be `<CATALINA_HOME>/conf/`

9. Create a `krb5.ini` file in `<CATALINA_HOME>/conf/`. The contents of the file should look like:

```
[libdefaults]
default_realm=REALM
default_keytab_name="FILE:<CATALINA_HOME>/conf/tomcat.keytab"
default_tkt_enctypes=aes256-cts-hmac-sha1-96,aes128-cts-hmac-sha1-96
default_tgs_enctypes=aes256-cts-hmac-sha1-96,aes128-cts-hmac-sha1-96
forwardable=true

[realms]
REALM={
 kdc=DOMAIN_CONTROLLER_FQDN:88
}

[domain_realm]
yourdomain.com=REALM
*.yourdomain.com=REALM
```

Adjust the values according to your configuration but keep the `FILE:` prefix for the keytab name. If you want to use a different location or file name for this file you can do

so by defining the following Java system property in
`<CATALINA_HOME>/conf/system.properties`:

```
java.security.krb5.conf=PATH_TO_KRB5_CONF
```

10. Create or edit the file `<CATALINA_HOME>/conf/jaas.conf`. The contents of the file should look like:

```
com.sun.security.jgss.krb5.accept {
  com.sun.security.auth.module.Krb5LoginModule
  required
  doNotPrompt=true
  principal="HTTP/TOMCAT_FQDN@REALM"
  keyTab="<CATALINA_HOME>/conf/tomcat.keytab"
  storeKey=true
  useKeyTab=true
  useTicketCache=true
  isInitiator=true
  refreshKrb5Config=true
  moduleBanner=true
  storePass=true;
};
com.sun.security.jgss.krb5.initiate {
  com.sun.security.auth.module.Krb5LoginModule
  required
  doNotPrompt=true
  principal="HTTP/TOMCAT_FQDN@REALM"
  keyTab="<CATALINA_HOME>/conf/tomcat.keytab"
  storeKey=true
  useKeyTab=true
  useTicketCache=true
  isInitiator=true
  refreshKrb5Config=true
  moduleBanner=true
  storePass=true;
};
```

Adjust the values according to your configuration. Note that the location to the keytab file must be given as an absolute path and contain forward slashes, even on Windows.

If you want to use a different location or file name for the `jaas.conf` you can do so by defining the following Java system property in
`<CATALINA_HOME>/conf/system.properties`:

```
java.security.auth.login.conf=PATH_TO_JAAS_CONF
```

In Kerberos documentation this file is often referred to as the `login.conf`.

11. Add the following property to the list of JVM system properties at startup. They can be defined in `<CATALINA_HOME>/conf/system.properties`:

```
javax.security.auth.useSubjectCredsOnly=false
```

12. Configure the `KNIMEServerAuthenticator` valve:

   a. Navigate to `<CATALINA_HOME>/conf/Catalina/localhost/`

   b. Edit the `knime.xml` file (the name of the file is equal to the context root that was set in the KNIME Server installer, the default is `knime`, if the `knime.war` file was renamed to `renamed.war`, the xml file will be called `renamed.xml`)

   c. Find the line

```
<Valve
className="com.knime.enterprise.tomcat.authenticator.KnimeServerAuthenticator
" enableSpnego="false"
 basicAuthPaths="/rest,/webservices" formAuthPaths="/" />
```

   d. Change it to

```
<Valve
className="com.knime.enterprise.tomcat.authenticator.KnimeServerAuthenticator
" enableSpnego="true"
basicAuthPaths="/rest,/webservices" />
```

   e. By default, the REST and SOAP webservices are set up to use basic HTTP authentication. If you want to use Single-Sign-On also for the REST and/or SOAP webservices, e.g. if you are using a REST client that supports Kerberos, adjust the `basicAuthPaths` attribute accordingly. It is a comma separated list of paths overwriting the default authentication method. Deleting the attribute enables Kerberos for all services.

   For example if REST is supposed to be used with Single-Sign-On the attribute would look like this: `basicAuthPaths="/webservices"`

13. Modify the `server.xml` and adjust the `JNDIRealm` settings to connect to your LDAP. If you have successfully tested your setup in step 3, it is sufficient to remove the `connectionName` and `connectionPassword` attributes.

   Please note that with Kerberos the `connectionName` and `connectionPassword` attributes

are ignored. Also the use of the `userPattern` is not supported by Tomcat when using Kerberos. Use `userBase` in combination with `userSearch` instead.

The realm definition could look like this:

```
<Realm className="org.apache.catalina.realm.JNDIRealm"
    connectionURL="ldap://dc.domain.com:3268"
    userSubtree="true"
    userBase="cn=Users,dc=domain,dc=com"
    userSearch="(sAMAccountName={0})"
    userRoleName="memberOf"
    roleBase="cn=Users,dc=domain,dc=com"
    roleName="cn"
    roleSearch="(member={0})"
    roleSubtree="true"
    roleNested="true"/>
```

If you are using Kerberos in a combined realm, make sure the JNDIRealm connecting to your LDAP is **first** in the list of realms.

14. Restart KNIME Server for the changes to take effect. Inspect the log files in `<CATALINA_HOME>/logs` to make sure that there are no error messages relating to your changes.

## Client Configuration

Client configuration requires two steps:

1. The client machine needs to be part of the domain, and the end user logged into that domain.
2. All browsers used by the client need to have Kerberos authentication enabled. The following sections describe how to Internet Explorer and Firefox for Kerberos authentication.

### Enabling Kerberos Authentication in Internet Explorer

1. Open the "Internet Options" menu and browse to the "Advanced" tab.
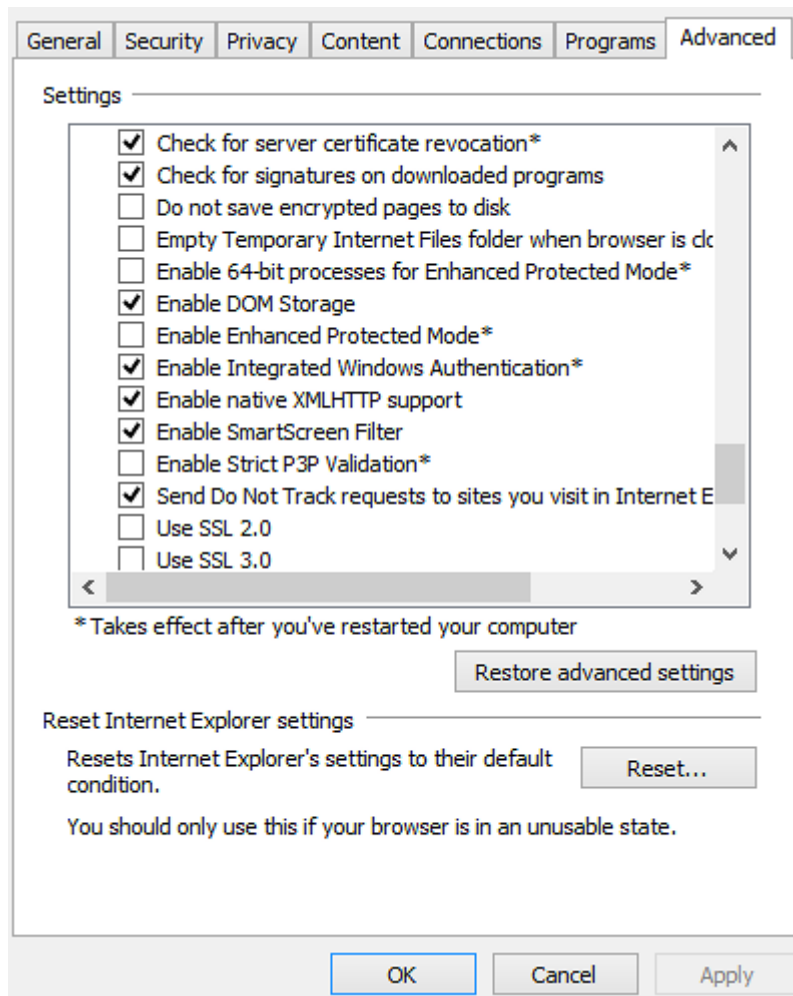
The setting "Enable Integrated Windows Authentication" needs to be checked.

2. Browse to the "Security" tab, select "Local Intranet" and click on the "Sites" button.
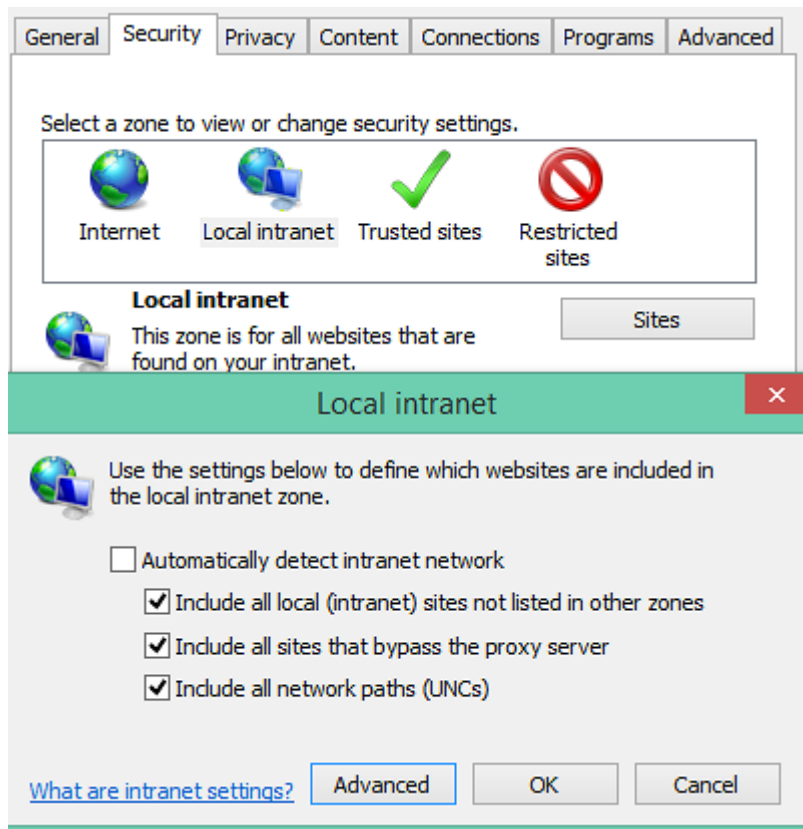
3. Click on "Advanced" and add the URL of KNIME Server to the list of websites in the zone.

4. Click on "Custom Level" and check that in *Local Intranet Security Level → User Authentication* is set to "Automatic logon only in Intranet zone"
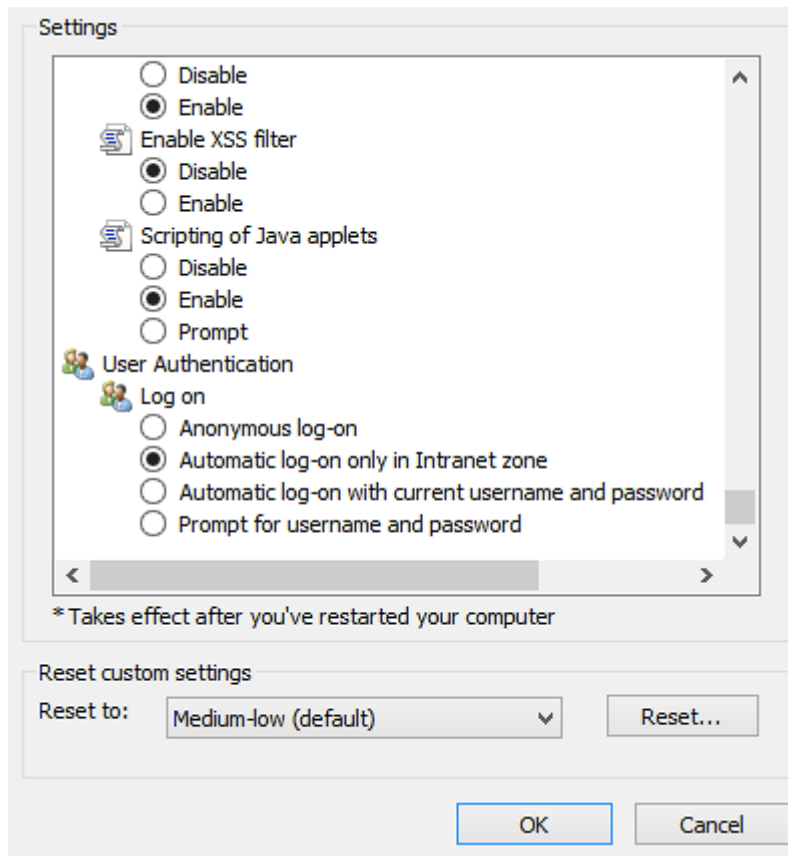
5. It might be necessary to also add KNIME Server to the list of trusted sites. To do so, go to "Trusted Sites" and click on the "Sites" button. Add the URL of KNIME Server to the list of websites in the zone.

6. Check that the *Trusted Sites Security Level → User Authentication* is set to "Automatic log-on with current username and password".
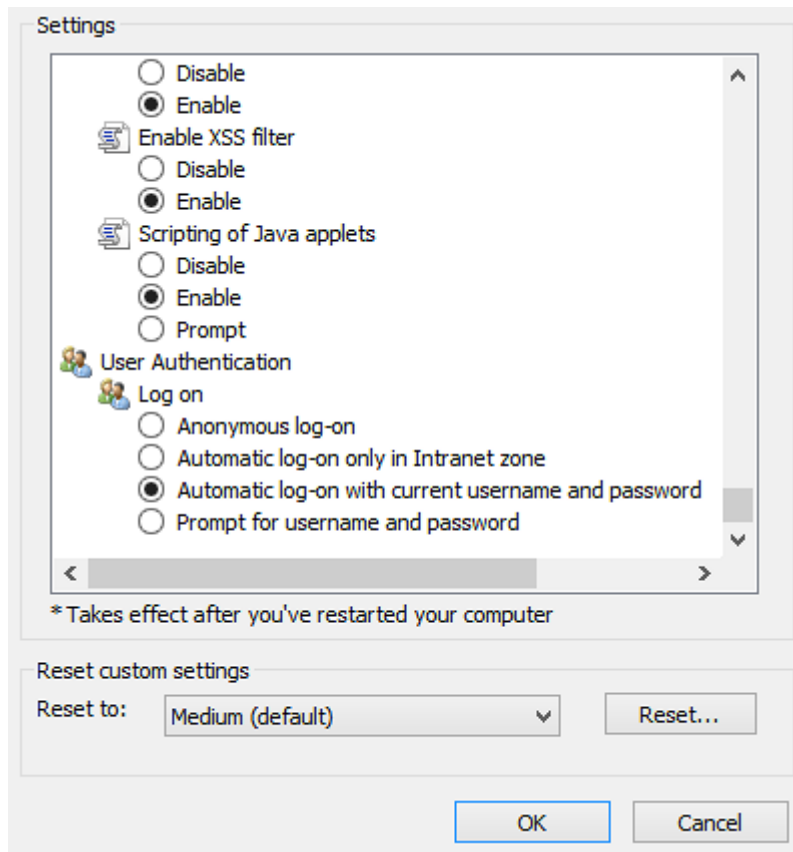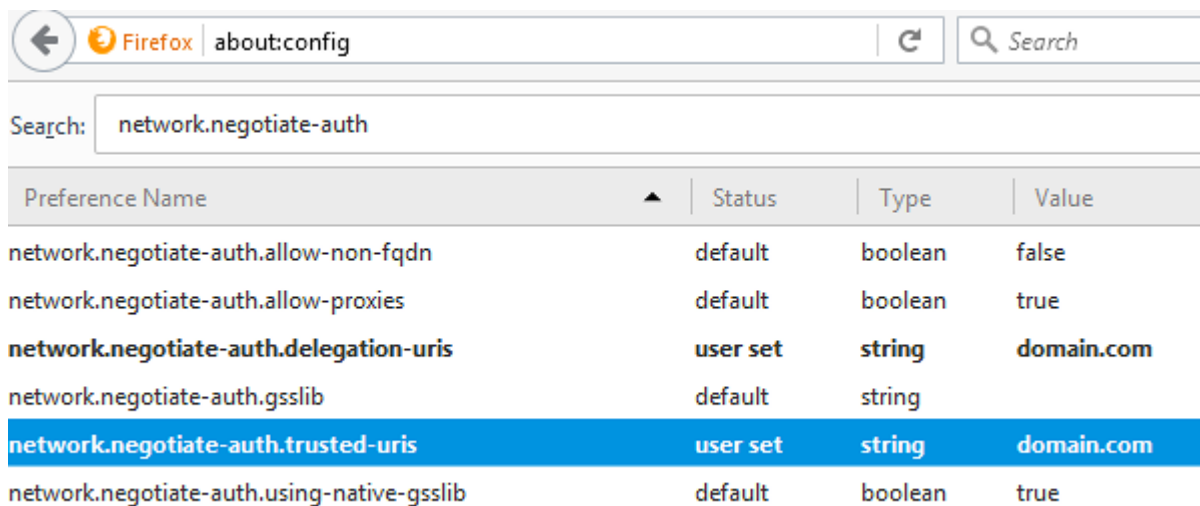
Enable Kerberos Authentication in Firefox

1. Start Firefox and type `about:config` in the address bar.

2. Ignore the warning by clicking on the "I'll be careful, I promise!" button.
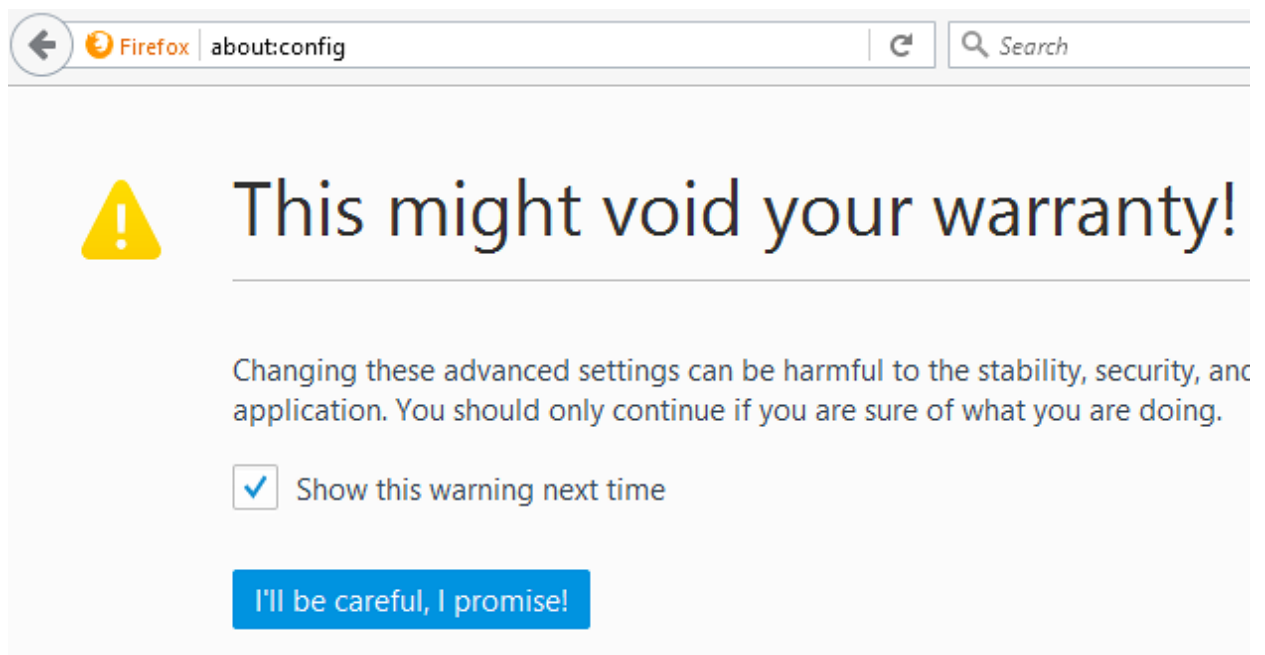


3. Find the appropriate settings by typing `network.negotiate-auth` in the search field.

Change the `network.negotiate-auth.delegation-uris` and `network.negotiate-auth.trusted-uris` to contain the URL of KNIME Server. It might be enough to just enter your domain.

## Troubleshooting

A Kerberos setup is usually very complex and needs precise configuration. Error messages are often times cryptic. To debug a Kerberos setup it is very helpful to enable additional logging for the authentication in the Tomcat server. To do so you can configure a few things.

1. To enable logging in the Krb5 modules, add or enable the following two lines in both section of the `jaas.conf` (or `login.conf` in `<CATALINA_HOME>/conf`):

   ```
   debug=true
   moduleBanner=true
   ```

   Note that the debug output is only printed to console.

2. To increase the debug output of the Kerberos implementation in Java add the following system property on startup (can be done in `system.properties` file in `<CATALINA_HOME>/conf`):

   ```
   -Dsun.security.krb5.debug=true
   ```

3. Add debugging for authentication and realm modules by adding to the `logging.properties` file in `<CATALINA_HOME>/conf`. For clarity all authentication output can be logged into a separate file.

```
[...]

4auth.org.apache.juli.FileHandler.level = FINE
4auth.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
4auth.org.apache.juli.FileHandler.prefix = auth.

[...]

org.apache.catalina.realm.level = ALL
org.apache.catalina.realm.handlers = 4auth.org.apache.juli.FileHandler

org.apache.catalina.authenticator.level = ALL
org.apache.catalina.authenticator.handlers = 4auth.org.apache.juli.FileHandler

com.knime.enterprise.tomcat.handlers = 4auth.org.apache.juli.FileHandler
com.knime.enterprise.tomcat.level = DEBUG

org.apache.juli.logging.UserDataHelper.CONFIG = INFO_ALL

org.apache.coyote.http11.level = DEBUG
org.apache.coyote.http11.handlers = 4auth.org.apache.juli.FileHandler
```

# Dynamic profiles for server-managed customizations

As mentioned in the KNIME Server Administration Guide it is possible to write a custom profile provider which selects the server and the list of profiles dynamically. This custom provider must be an implementation of the `org.knime.product.profiles.IProfileProvider` interface which is contained in the `org.knime.product` plug-in. The implementation of this interface must not make use of any classes that trigger reading preferences, otherwise the default preferences cannot be changed any more. This includes usage of commonly used KNIME classes such as `KNIMEConstants` or `NodeLogger`.

Therefore we suggest to create a new plug-in that only has a dependency to `org.knime.product` (for the `IProfileProvider` interface) and doesn't use any other KNIME classes otherwise. An exception are classes from the `org.knime.core.util` plug-in because it doesn't use any preferences (and never will be). Other than that, the implementation is straight-forward. The class `org.knime.product.profiles.ExampleProfileProvider` contains a minimal example of a custom profile provider that you can use as a starting point. Don't forget to register your implementation at the extension point `org.knime.product.profileProvider`.

If you have any questions implementing a custom profile provider, don't hesitate to contact us.

# OpenID Connect Authentication

The KNIME Server can be configured to use an OpenID Connect enabled Identity Provider for authentication.

When the feature is enabled and configured an authenticated user will be mapped to a KNIME Server user.

> **i** Please note that only the Authorization Code Flow is supported and not the ID Token Flow.

## Authentication Valve configuration for OpenID Connect (OAuth)

To enable OAuth authentication for the KNIME Server, the file `<apache-tomcat>/conf/Catalina/localhost/knime.xml` has to be edited.

It is possible to configure the server such that both OAuth and basic authentication, using credentials, can be used at the same time. For the configuration the following parameters can be added to the authentication valve's definition:

---

`enableOAuth="<true|false>"`

Enables OAuth authentication.
The default is false (OAuth authentication disabled).

---

`enableBasicAuthWithOAuth="<true|false>"`

Enables basic authentication along side OAuth if OAuth is enabled. The default is false (basic authentication disabled when using OAuth). With this option enabled, REST clients and the KNIME Analytics platform can still authenticate using the user's credentials (username and password). Login to the WebPortal will only be possible using OAuth through the Identity Provider.

---

`oAuthConfigurationPath="<Path to configuration file>"`

The path to the configuration file.
The recommended location is `<apache-tomcat>/conf/Catalina/localhost/knime-oidc-config.json`.

---

The Valve entry should look similar to this:

```
<Valve className="com.knime.enterprise.tomcat.authenticator.KnimeServerAuthenticator"
  enableSpnego="false" basicAuthPaths="/rest" formAuthPaths="/"
  secretKey="someSecreKey" enableOAuth="true" enableBasicAuthWithOAuth="false"
  oAuthConfigurationPath="/path/to/conf/Catalina/localhost/knime-oidc-config.json"/>
```

The provider-specific configuration is done by creating a `knime-oidc-config.json` file and placing it in according to the path configured in `<apache-tomcat>/conf/Catalina/localhost/knime.xml`, where `<apache-tomcat>/conf/Catalina/localhost/knime-oidc-config.json` is the recommended location.

Here is an example of such a file, the parameters are explained below:

```
{
  "identity-provider-name": "Some Identity Provider",
  "auth-server-url": "https://identitiy.provider/",
  "resource": "client-id",
  "credentials": {
    "secret": "client-secret"
  },
  "additional-authorization-endpoint-parameters": "&additional-parameter=some-
value&some-other-parameter=some-value",
  "additional-scopes": "additional-scope another-scope",
  "principal-attribute": "claim-used-for-principal-mapping"
}
```

> ℹ️ Please be aware that this is a `JSON` file so it should comply the `JSON` standard, e.g. comma after each entry except for the last one.

The authenticator is capable of discovering the necessary OpenID Connect endpoints automatically. It does so by using the `auth-server-url` and inspecting the ".well-known/openid-configuration" endpoint if it is available. If the discovery fails the endpoints must be set explicitly.

`"identity-provider-name": "<identity-provider-name>"`

> The name of the Identity provider to be displayed on the login landing page as:
> "Login using <identity-provider-name>"

`"auth-server-url": "<auth-server-url>"`

> The base URL for the Identity Provider's OpenID Connect endpoint.
> This endpoint is used for automatic endpoint discovery, according to OpenID Connect discovery.

`"authorization-endpoint": "<authorization-endpoint>"`

(OPTIONAL) The Identity Provider's authorization endpoint.
Must be set explicitly if automatic endpoint discovery fails.

`"token-endpoint": "<token-endpoint>"`

(OPTIONAL) The Identity Provider's token endpoint.
Must be set explicitly if automatic endpoint discovery fails.

`"jwks-endpoint": "<jwks-endpoint>"`

(OPTIONAL) The JWKS endpoint to verify JWT Tokens.
Must be set explicitly if automatic endpoint discovery fails.

`"userinfo-endpoint": "<userinfo-endpoint>"`

(OPTIONAL) The Identity Provider's userinfo endpoint.
Must be set explicitly if automatic endpoint discovery fails.

`"resource": "<client-id>"`

The application's client ID.

-

`"credentials": {"secret": "<client-secret>"}`

(OPTIONAL) The client secret if it must be set for the Identity Provider.

-

`"public-client": "<true|false>"`

(OPTIONAL) If no client credentials are needed, set to "true".

-

`"additional-authorization-endpoint-parameters": "&<additionalQueryParameters>"`

(OPTIONAL) Additional parameters used when calling the Identity Provider's
authorization endpoint.

-

`"additional-scopes": "<scope> <scope>"`

Space-separated list of additional scopes that should be requested when calling the Identity Provider's authorization endpoint. The *openid* scope is always used when talking to the authorization endpoint.

`"principal-attribute": "<claim>"`

The claim that should be used to map the authenticated users to the tomcat realm. This claim is also used as the account name for the users accessing the KNIME Server. By default this will use the *sub* claim. The *nickname* claim could be used for example, in this case, make sure that the corresponding scope is requested, when calling the authorization endpoint.

`"minimal-access-token-parsing": "<true|false>"`

If the access token can't be verified because one of its claims is malformed or does not correspond to the specification (see https://openid.net/specs/openid-connect-core-1_0.html), parsing the access token can be held to a minimum, so that the token's signature and issuer can still be verified.

`"allow-opaque-access-token": "<true|false>"`

If the Identity Provider can't provide access tokens as JWT the opaque token cannot be verified by inspecting its claims nor by verifying its signature. The verification is then left to the Identity Provider, by calling the userinfo endpoint. This is not recommended, but it is the only way to enable compatibility for Identity Providers that do not provide JWT access tokens.

`"treat-access-token-as-opaque": "<true|false>"`

If the access token's signature can't be verified, because it does not follow the specification, for example because it contains headers that need to be handled specially for signature verification, the tokens can be treated as opaque. In that case the access tokens are not parsed and their signature and issuer are not verified. The verification is then left to the Identity Provider, by calling the userinfo endpoint. This is not recommended, but can help as a workaround for incompatible Identity Providers, that do not conform to the OIDC specification. If this option is set to "true", both the minimal-access-token-parsing, as well as the allow-opaque-access-token option will be ignored.

```
"principal-attribute-to-username-regex": "<regex>"
```

> (OPTIONAL) The principal-attribute can be modified using a regular expression. For example if the email claim is configured as principal-attribute, an email like *john.doe@company.com* could be mapped to *john.doe*, using the regex "@company.com". The first substring of the principal-attribute that matches the regular expression will be removed.

```
"redirect-rewrite-rules": "<list-of-rules"
```

> (OPTIONAL) Might be needed, specifies the Redirect URI rewrite rules. This is an object notation, the key is a regular expression to which the Redirect URI is to be matched and the value is the replacement String.

```
"perform-direct-redirect": "<true|false>"
```

> (OPTIONAL) When this option is set to "true", the KNIME Webportal landing page will not be shown. Instead the user will be directly forwarded to the identity provider for authentication. The user will still be presented with the landing if the login fails.

```
"proxy-url": "<proxy-url>"
```

> (OPTIONAL) The proxy configuration that should be used to talk to the Identity Provider.
> The configuration parameter `<proxy-url>` takes the form *http://user:password@hostname:port*.

## User and group management

When the feature is enabled, users will be mapped to an account in KNIME Server using a configurable claim from the userinfo endpoint. For example if the claim *nickname* is chosen, let it be "john.doe", the user will be mapped to an internal user with the username "john.doe".

### Restricting login groups

By default all authenticated users will be able to login in to the KNIME Server. If access should be restricted, allowed login groups can be defined in the KNIME Server Configuration File, this way users that are not assigned to any group cannot log in to the server.

Adding the following configuration option, will restrict the login to the specified groups:

```
com.knime.server.login.allowed_groups =<group>,<group>,···
```

> Defines the groups that are allowed to log in to the server.
> Default value allows users from all groups.

## Tomcat group management

By default the group management is handled by the tomcat realm. This way the tomcat realms can still be configured according to the User Authentication section in the KNIME Server Administration Guide. Note that the password set for the user in the tomcat realm does not matter for the OAuth authentication. As long as the defined claim matches with a username in the database the user will be logged in with the assigned groups.

## LDAP configuration

Using the JNDIRealm for LDAP is also a valid configuration, as long as the principal-attribute which maps the user name, can be mapped to an entry in LDAP. A valid configuration for the user/group lookup, with the principal-attribute email, could look like this:

```
<Realm className="org.apache.catalina.realm.CombinedRealm">
  <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                resourceName="UserDatabase" />
  <Realm className="org.apache.catalina.realm.JNDIRealm"
                connectionURL="ldap://ldap.hostname:389"
                userBase = "ou=people, dc=company, dc=com"
                userSearch = "(email={0})"
                roleBase="ou=groups,dc=company,dc=com" roleName="cn"
                roleSearch="(member={0})" />
</Realm>
```

Please refer to the KNIME Server Advanced Setup Guide for more information on setting up LDAP.

## Group mapping claim

The authenticator can also be configured to map a custom claim retrieved from the userinfo endpoint to user groups. This could be a claim called 'groups', for example. The group retrieval can enabled by defining the claim in the knime-oidc-config.json, with the following parameter:

```
group-mapping-claim="<group-claim>"
```
> The name of the claim that provides an array of groups for the user at the userinfo endpoint. Make sure that the scopes are defined appropriately, so that the claim can be retrieved.

If the group mapping claim is enabled, the tomcat realm is not used.

Azure Active Directory and Graph API

In order to enable group management through Azure Active directory, the KNIME Server authenticator can be configured to call the Microsoft Graph API for group retrieval. The following table describes the available configuration parameters, which can be set in the knime-oidc-config.json.

```
graph-api-group-information="<true|false>"
```
> If this is set to "true", the authenticator will call the Graph API to retrieve the user groups.
>
> -

```
graph-api-use-display-name="<true|false>"
```
> If this is set to "true", the group's displayName is used as group name, if it is available. If the displayName cannot be retrieved the group ID will be used as group name. If the parameter is set to "false", the group id will be used in any case.

It is required that the Application has the Directory.Read.ALL API permission with admin consent, so that the groups can be retrieved through the Graph API.

## KNIME Server Client configuration for the KNIME Analytics Platform

In order to enable OAuth authentication for clients of a KNIME Server configured for OpenID Connect, some additional configuration has to be added to the <knime-server-repository>/config/knime-server.config file.

By default, the endpoints, client id and client secret are configured using the knime-oidc-config.json.

The following show what the additional configuration could look like, the configuration parameters are described below:

```
com.knime.server.authentication.oauth.redirect_ports=8888,8881,8882
com.knime.server.authentication.types=OAuth,Credentials
com.knime.server.authentication.preferredType=OAuth
com.knime.server.authentication.oauth.token_refresh_rate=5m
com.knime.server.authentication.oauth.allowed_clock_skew=30s
```

`com.knime.server.authentication.types=<type>,<type>`

Comma-separated list of authentication types. Possible values are "OAuth" and "Credentials". This should match the authentication valve configuration in `<apache-tomcat>/conf/Catalina/localhost/knime.xml`.

`com.knime.server.authentication.preferredType=<type>`

The preferred authentication type that should be used by KNIME Analytics Platform. The preferred type will be pre-selected when adding KNIME Server as a new mount point in KNIME Analytics Platform. Possible values are "OAuth" and "Credentials". This should match the authentication types defined in `com.knime.server.authentication.type`.

`com.knime.server.authentication.oauth.redirect_ports=<port>,<port>,<port>`

A comma-separated list of ports that should be used for the authorization redirect. If no list is given a random port will be chosen for each authorization. Some Identity Providers might not support wildcards, in this case a list of configured ports should be provided here.

`com.knime.server.authentication.oauth.token_refresh_rate=<duration with unit, e.g. 5m, 30m or 2h>`

(OPTIONAL) If the access token is opaque, the expiry date of the access token cannot be determined, so a refresh rate can be introduced, to refresh the token.

`com.knime.server.authentication.oauth.allowed_clock_skew=<duration with unit, e.g. 30s or 1m>`

The allowed clock skew used when the access token is parsed in KNIME Analytics Platform. This can be used to handle potential clock skew between the Identity Provider and KNIME Analytics Platform.

`com.knime.server.authentication.oauth.client_secret=<value>`

> (OPTIONAL) Only use this option if it is required by the Identity Provider. The client secret that has to be used by KNIME Analytics Platform to be able to authenticate against the Identity Provider.

`com.knime.server.authentication.oauth.authorization_endpoint=<authorization-endpoint>`

> (OPTIONAL, provided from knime-odic-config.json)

The Identitiy Provider's authorization endpoint.

`com.knime.server.authentication.oauth.token_endpoint=<token-endpoint>`

> (OPTIONAL, provided from knime-odic-config.json)

The Identitiy Provider's token endpoint.

`com.knime.server.authentication.oauth.client_id=<client-id>`

> (OPTIONAL, provided from knime-odic-config.json)

The application's client ID.

`com.knime.server.authentication.oauth.scope=<scope>,<scope>,<scope>`

> (OPTIONAL, provided from knime-odic-config.json)

Comma-separated list of additional scopes that should be requested when calling the Identity Providers's authorization endpoint. The scopes should match the scopes in the knime-oidc-config.json on the server side. The **openid** scope should be provided in any case, along with the additional scopes also defined in the knime-oidc-config.json.

`com.knime.server.authentication.oauth.additional_query_params=<query-string>`

> (OPTIONAL, provided from knime-odic-config.json)

Additional parameters that should be used when calling the Identity Provider's authorization endpoint, provided as a query string.

The callback URL for the KNIME Analytics Platform is

```
http://127.0.0.1:<port>/oauthredirectlistener
```

where <port> depends on the configured redirect port(s).

## Debugging OIDC Authentication

In order to enable logging, to debug authentication problems, some changes have to be applied to the `<apache-tomcat>/conf/logging.properties`.

Firstly, the handler **4auth.org.apache.juli.FileHandler** has to be added to the list of handlers:

```
handlers = 4auth.org.apache.juli.FileHandler,....
```

Secondly, the log level for this handler should be set to **ALL**:

```
4auth.org.apache.juli.FileHandler.level = ALL
4auth.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
4auth.org.apache.juli.FileHandler.prefix = auth.
4auth.org.apache.juli.FileHandler.maxDays = 90
```

Finally, the block concerning authentication should be uncommented:

```
#org.apache.catalina.realm.level = ALL
#org.apache.catalina.realm.handlers = 4auth.org.apache.juli.FileHandler
#org.apache.catalina.authenticator.level = ALL
#org.apache.catalina.authenticator.handlers = 4auth.org.apache.juli.FileHandler
com.knime.enterprise.tomcat.handlers = 4auth.org.apache.juli.FileHandler
com.knime.enterprise.tomcat.level = ALL
#org.apache.juli.logging.UserDataHelper.CONFIG = INFO_ALL
#org.apache.coyote.http11.level = DEBUG
#org.apache.coyote.http11.handlers = 4auth.org.apache.juli.FileHandler
```

With this logging configuration the log output for authentication can be found under `<apache-tomcat>/logs/auth.yyyy-mm-dd.log`

# Using your own Tomcat installation

KNIME Server is based on Apache Tomcat therefore it is possible to use your own Apache Tomcat installation instead of relying on the version packaged with KNIME Server. This can be useful if your corporate IT policies require you to use a certain version of Apache Tomcat that is different from what the KNIME Server Installer provides.

While it is technically possible to run KNIME Server inside an existing Apache Tomcat installation that already hosts other web applications we recommend using a dedicated installation. The reason is that certain modifications to the Apache Tomcat installation itself are required for proper function of KNIME Server. These modifications may have side effects to other web applications running in the same installation.

These are the steps to install KNIME Server within a custom Apache Tomcat:

1. Install KNIME Server using the KNIME Server Installer.

2. Download and extract Apache Tomcat 9.0.x in the desired version. Only this version of Apache Tomcat is supported.

3. Copy the following files from the Apache Tomcat installation created by the KNIME Server Installer into your custom Apache Tomcat installation:

   ◦ `<apache-tomcat>/conf/server.xml`

   ◦ `<apache-tomcat>/conf/userconf.mv.db`

   ◦ `<apache-tomcat>/conf/Catalina/localhost/knime.xml`

   ◦ `<apache-tomcat>/lib/h2-xxx.jar`

   ◦ `<apache-tomcat>/lib/knime-tomcat.jar`

   ◦ `<apache-tomcat>/webapps/knime.war`

4. Configure and customize the installation as described in the KNIME Server Administration Guide

If you are already using KNIME Server you can copy the same files from this installation plus any additional files you created when customizing your installation (such as SSL certificates or OIDC configuration files). Make sure that you adapt your systemd/Windows services to start the new installation instead of the old one.

KNIME AG
Talacker 50
8001 Zurich, Switzerland
www.knime.com
info@knime.com