Install and configure LDAP

Installing slapd (the Stand-alone LDAP Daemon) creates a minimal working configuration with a top level entry, and an administrator's Distinguished Name (DN).

In particular, it creates a database instance that you can use to store your data. However, the **suffix** (or **base DN**) of this instance will be determined from the domain name of the host. If you want something different, you can change it right after the installation (before it contains any useful data).

Note:
This guide will use a database suffix of `dc=example,dc=com`. You can change this to match your particular setup.

Install slapd

You can install the server and the main command line utilities with the following command:

```
sudo apt install slapd ldap-utils
```

Change the instance suffix (optional)

If you want to change your Directory Information Tree (DIT) suffix, now would be a good time since changing it discards your existing one. To change the suffix, run the following command:

```
sudo dpkg-reconfigure slapd
```

To switch your DIT suffix to `dc=example,dc=com`, for example, so you can follow this guide more closely, answer `example.com` when asked about the DNS domain name.

Throughout this guide we will issue many commands with the LDAP utilities. To save some typing, we can configure the OpenLDAP libraries with certain defaults in `/etc/ldap/ldap.conf` (adjust these entries for your server name and directory suffix):

```
BASE dc=example,dc=com
URI ldap://ldap01.example.com
```

Configuration options

`slapd` is designed to be configured within the service itself by dedicating a separate DIT for that purpose. This allows for dynamic configuration of `slapd` without needing

to restart the service or edit config files. This configuration database consists of a collection of text-based LDIF files located under `/etc/ldap/slapd.d`, but these should never be edited directly. This way of working is known by several names: the "slapd-config" method, the "Real Time Configuration (RTC)" method, or the "cn=config" method. You can still use the traditional flat-file method (`slapd.conf`) but that will not be covered in this guide.

Right after installation, you will get two databases, or suffixes: one for your data, which is based on your host's domain (`dc=example,dc=com`), and one for your configuration, with its root at `cn=config`. To change the data on each we need different credentials and access methods:

- `dc=example,dc=com`
  The administrative user for this suffix is `cn=admin,dc=example,dc=com` and its password is the one selected during the installation of the `slapd` package.
- `cn=config`
  The configuration of `slapd` itself is stored under this suffix. Changes to it can be made by the special
  DN `gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth`. This is how the local system's root user (`uid=0/gid=0`) is seen by the directory when using SASL EXTERNAL authentication through the `ldapi:///` transport via
  the `/run/slapd/ldapi` Unix socket. Essentially what this means is that only the local root user can update the `cn=config` database. More details later.

Example `slapd-config` DIT

This is what the `slapd-config` DIT looks like via the LDAP protocol (listing only the DNs):

```
$ sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b cn=config dn

dn: cn=config
dn: cn=module{0},cn=config
dn: cn=schema,cn=config
dn: cn={0}core,cn=schema,cn=config
dn: cn={1}cosine,cn=schema,cn=config
dn: cn={2}nis,cn=schema,cn=config
dn: cn={3}inetorgperson,cn=schema,cn=config
dn: olcDatabase={-1}frontend,cn=config
dn: olcDatabase={0}config,cn=config
dn: olcDatabase={1}mdb,cn=config
```

Where the entries mean the following:

- `cn=config`: Global settings
- `cn=module{0},cn=config`: A dynamically loaded module
- `cn=schema,cn=config`: Contains hard-coded system-level schema

- `cn={0}core,cn=schema,cn=config`: The hard-coded *core* schema
- `cn={1}cosine,cn=schema,cn=config`: The Cosine schema
- `cn={2}nis,cn=schema,cn=config`: The Network Information Services (NIS) schema
- `cn={3}inetorgperson,cn=schema,cn=config`: The InetOrgPerson schema
- `olcDatabase={-1}frontend,cn=config`: Frontend database, default settings for other databases
- `olcDatabase={0}config,cn=config`: `slapd` configuration database (`cn=config`)
- `olcDatabase={1}mdb,cn=config`: Your database instance (`dc=example,dc=com`)

Example `dc=example,dc=com` DIT

This is what the `dc=example,dc=com` DIT looks like:

```
$ ldapsearch -x -LLL -H ldap:/// -b dc=example,dc=com dn

dn: dc=example,dc=com
dn: cn=admin,dc=example,dc=com
```

Where the entries mean the following:

- `dc=example,dc=com`: Base of the DIT
- `cn=admin,dc=example,dc=com`: Administrator (rootDN) for this DIT (set up during package install)

Notice how we used two different authentication mechanisms:

- `-x`
  This is called a "simple bind", and is essentially a plain text authentication. Since no **Bind DN** was provided (via `-D`), this became an *anonymous* bind. Without `-x`, the default is to use a Simple Authentication Security Layer (SASL) bind.
- `-Y EXTERNAL`
  This is using a SASL bind (no `-x` was provided), and further specifying the `EXTERNAL` type. Together with `-H ldapi:///`, this uses a local UNIX socket connection.

In both cases we only got the results that the server access-control lists (ACLs) allowed us to see, based on who we are. A very handy tool to verify the authentication is `ldapwhoami`, which can be used as follows:

```
$ ldapwhoami -x

anonymous
```

```
$ ldapwhoami -x -D cn=admin,dc=example,dc=com -W

Enter LDAP Password:
dn:cn=admin,dc=example,dc=com
```

When you use simple bind (`-x`) and specify a Bind DN with `-D` as your authentication DN, the server will look for a `userPassword` attribute in the entry, and use that to verify the credentials. In this particular case above, we used the database **Root DN** entry, i.e., the actual administrator, and that is a special case whose password is set in the configuration when the package is installed.

Note:
A simple bind without some sort of transport security mechanism is **clear text**, meaning the credentials are transmitted in the clear. You should add Transport Layer Security (TLS) support to your OpenLDAP server as soon as possible.

Example SASL EXTERNAL

Here are the SASL EXTERNAL examples:

```
$ ldapwhoami -Y EXTERNAL -H ldapi:/// -Q

dn:gidNumber=1000+uidNumber=1000,cn=peercred,cn=external,cn=auth

$ sudo ldapwhoami -Y EXTERNAL -H ldapi:/// -Q

dn:gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
```

When using SASL EXTERNAL via the `ldapi:///` transport, the Bind DN becomes a combination of the `uid` and `gid` of the connecting user, followed by the suffix `cn=peercred,cn=external,cn=auth`. The server ACLs know about this, and grant the local root user complete write access to `cn=config` via the SASL mechanism.

Populate the directory

Let's introduce some content to our directory. We will add the following:

- A node called **People**, to store users
  - A user called **john**
- A node called **Groups**, to store groups
  - A group called **miners**

Create the following LDIF file and call it `add_content.ldif`:

```
dn: ou=People,dc=example,dc=com
objectClass: organizationalUnit
ou: People

dn: ou=Groups,dc=example,dc=com
objectClass: organizationalUnit
ou: Groups

dn: cn=miners,ou=Groups,dc=example,dc=com
objectClass: posixGroup
cn: miners
gidNumber: 5000

dn: uid=john,ou=People,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: john
sn: Doe
givenName: John
cn: John Doe
displayName: John Doe
uidNumber: 10000
gidNumber: 5000
userPassword: {CRYPT}x
gecos: John Doe
loginShell: /bin/bash
homeDirectory: /home/john
```

Note:
It's important that `uid` and `gid` values in your directory do not collide with local values. You can use high number ranges, such as starting at 5000 or even higher.

Add the content:

```
$ ldapadd -x -D cn=admin,dc=example,dc=com -W -f add_content.ldif

Enter LDAP Password: ********
adding new entry "ou=People,dc=example,dc=com"

adding new entry "ou=Groups,dc=example,dc=com"
adding new entry "cn=miners,ou=Groups,dc=example,dc=com"
adding new entry "uid=john,ou=People,dc=example,dc=com"
```

We can check that the information has been correctly added with the `ldapsearch` utility. For example, let's search for the "john" entry, and request the `cn` and `gidnumber` attributes:

```
$ ldapsearch -x -LLL -b dc=example,dc=com '(uid=john)' cn gidNumber

dn: uid=john,ou=People,dc=example,dc=com
cn: John Doe
gidNumber: 5000
```

Here we used an LDAP "filter": `(uid=john)`. LDAP filters are very flexible and can become complex. For example, to list the group names of which **john** is a member, we could use the filter:

```
(&(objectClass=posixGroup)(memberUid=john))
```

That is a logical "AND" between two attributes. Filters are very important in LDAP and mastering their syntax is extremely helpful. They are used for simple queries like this, but can also select what content is to be replicated to a secondary server, or even in complex ACLs.

Notice we set the `userPassword` field for the "john" entry to the cryptic value `{CRYPT}x`. This essentially is an invalid password, because no hashing will produce just `x`. It's a common pattern when adding a user entry without a default password. To change the password to something valid, you can now use `ldappasswd`:

```
$ ldappasswd -x -D cn=admin,dc=example,dc=com -W -S
uid=john,ou=people,dc=example,dc=com

New password:
Re-enter new password:
Enter LDAP Password:
```

How to Install Apache Directory Studio and Connect to an OpenLDAP Server

Apache Directory Studio is part of the Apache Directory project that strives to *increase LDAP awareness, comfort and adoption to bring forth what we call the Modern LDAP Renaissance*. The project includes:

- Apache Directory Server — an extensible and embeddable directory server

- Apache LDAP API — an enhanced LDAP API
- Apache Mavibot — a Multi Version Concurrency Control (MVCC) BTree
- Apache Kerby — a Java Kerberos binding
- Apache Fortress — a standards-based Access Management System

And, of course, the Apache Directory Studio, which is a tool intended to be used with any LDAP platform.

Installing Apache Directory Studio

1. Download the Apache Directory Studio tar file from the official Apache site
2. Save the file to the chosen location
3. Open a terminal window
4. Change into the chosen directory
5. Unpack the downloaded file with the command *tar xvzf ApacheDirectoryStudio-XXX.yyy.tar.gz* (where XXX is the release number and yyy is either 32 or 64 bit)
6. Change into the newly created *ApacheDirectoryStudio* directory with the command *cd ApacheDirectoryStudio*
7. Start the software with the command *./ApacheDirectoryStudio*

At this point, you should now see the Apache Directory Studio main window (Figure 2).

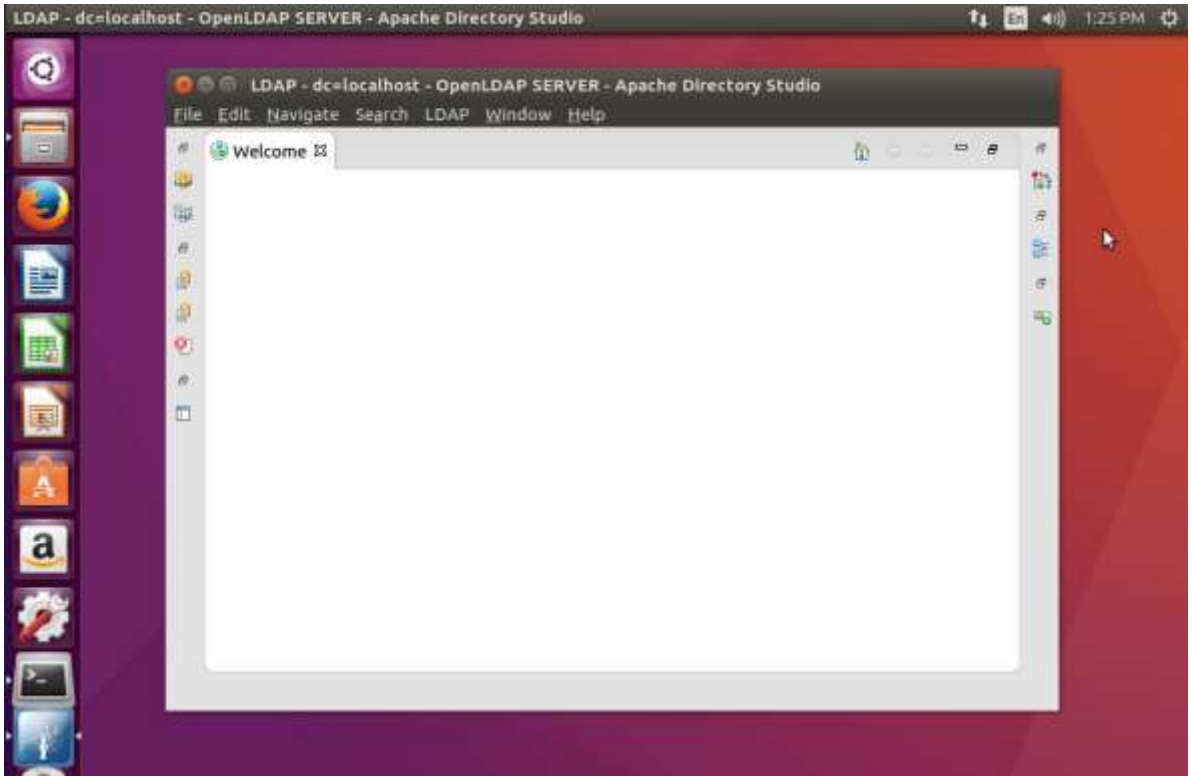

Figure 2: The Apache Directory Studio main window ready to work.

Connecting to an LDAP server

You are now ready to connect Apache Directory Studio to your LDAP server. Click File > New and then select LDAP Connection (Figure 3).

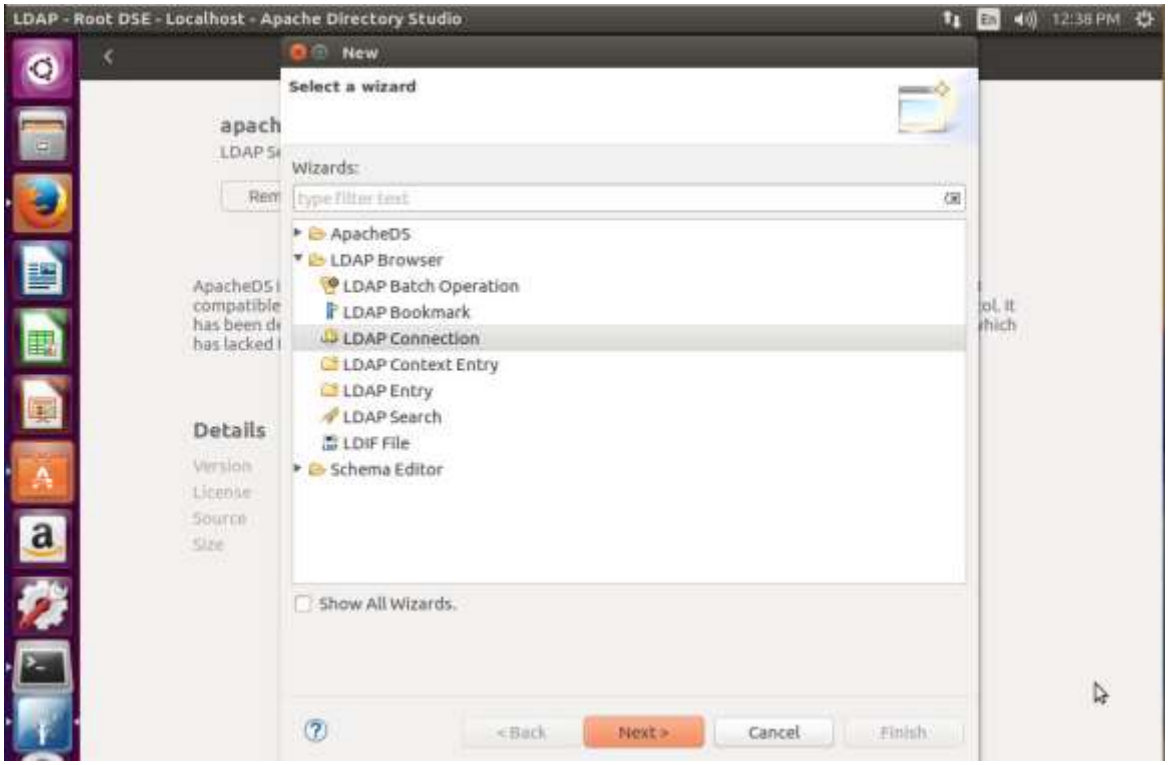

Figure 3: Starting the LDAP Connection wizard.

In the next window (Figure 4), you must enter the information for your LDAP server. Give it a name, enter the hostname (or IP address), port number, select the encryption method, and the provider. Once you've filled out that information, click Check Network Parameter to make sure everything is working properly.

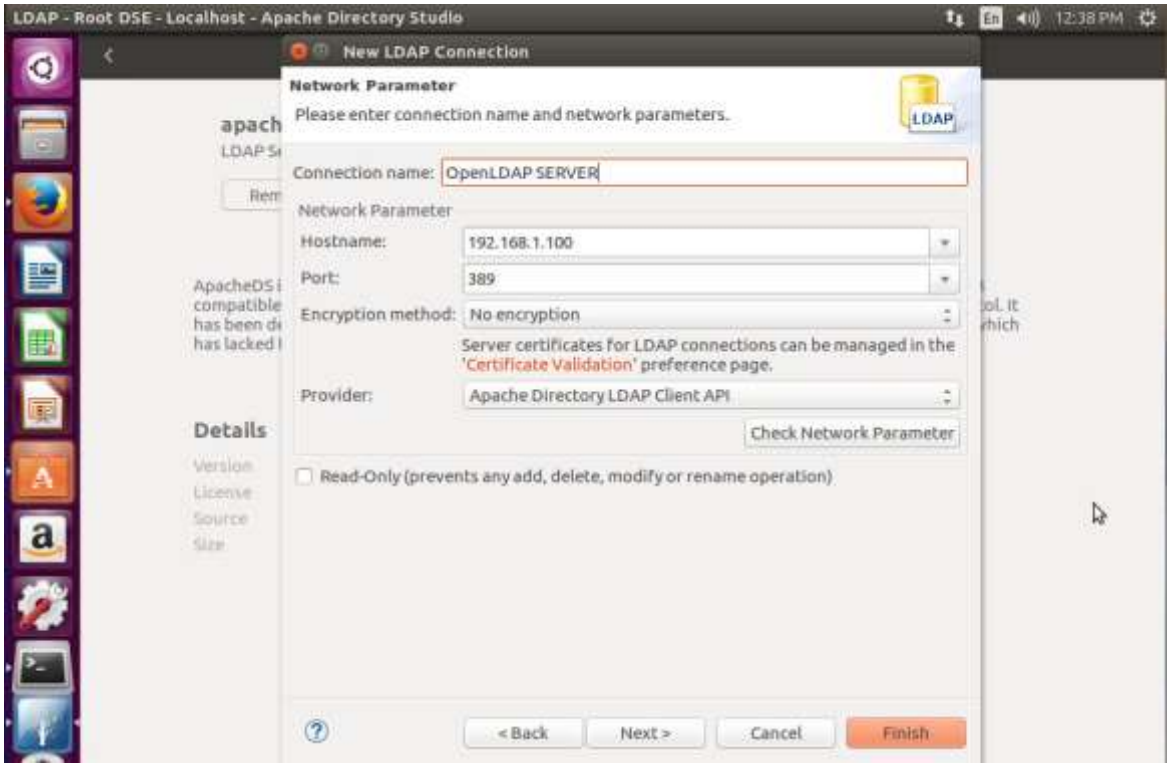

Figure 4: Setting up your LDAP configuration.

Click Next and you will then be required to fill out the authentication information for your connection (Figure 5). Select the Authentication Method, Bind DN or user, Bind password, and then click Check Authentication.

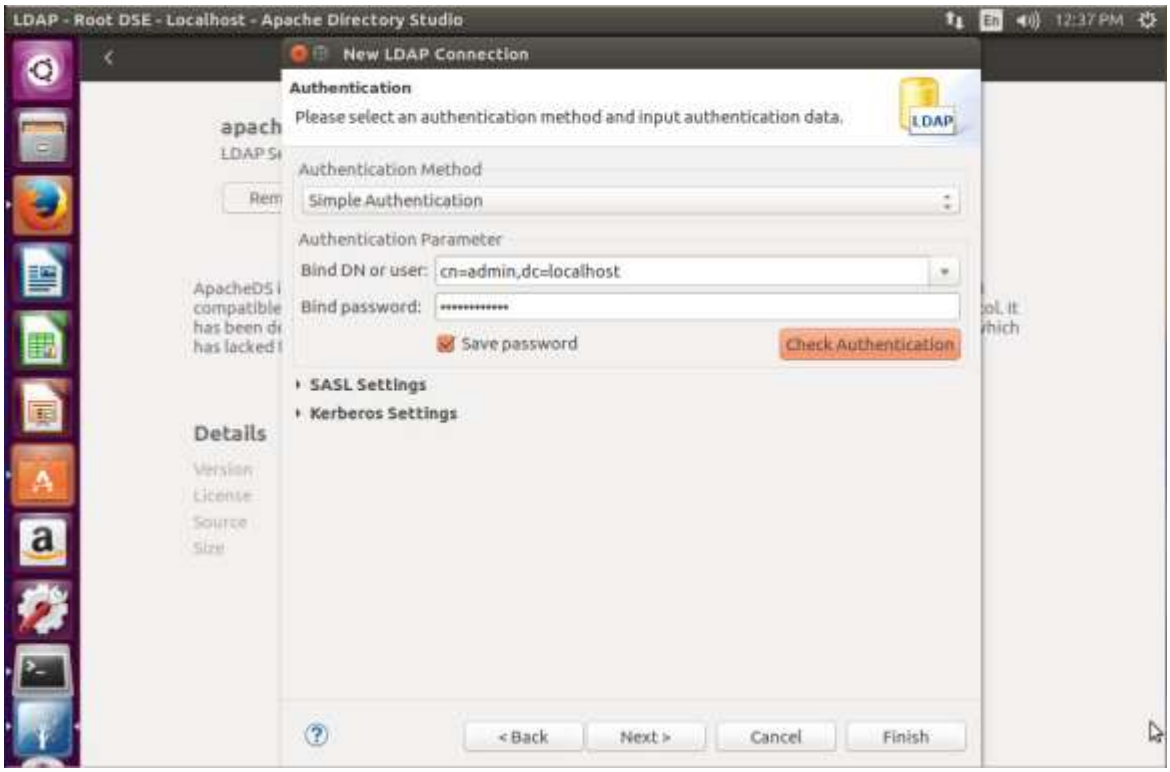

Figure 5: Filling out the authentication requirements for your LDAP server.

If your LDAP server requires SASL or Kerberos to be configured, expand those options and fill them out. Once you've completed this window, click Next.

In the next window (Figure 6), you can specify additional parameters for browsing your LDAP directory. As with many of the other options, these will depend upon your needs and how your LDAP server was configured.

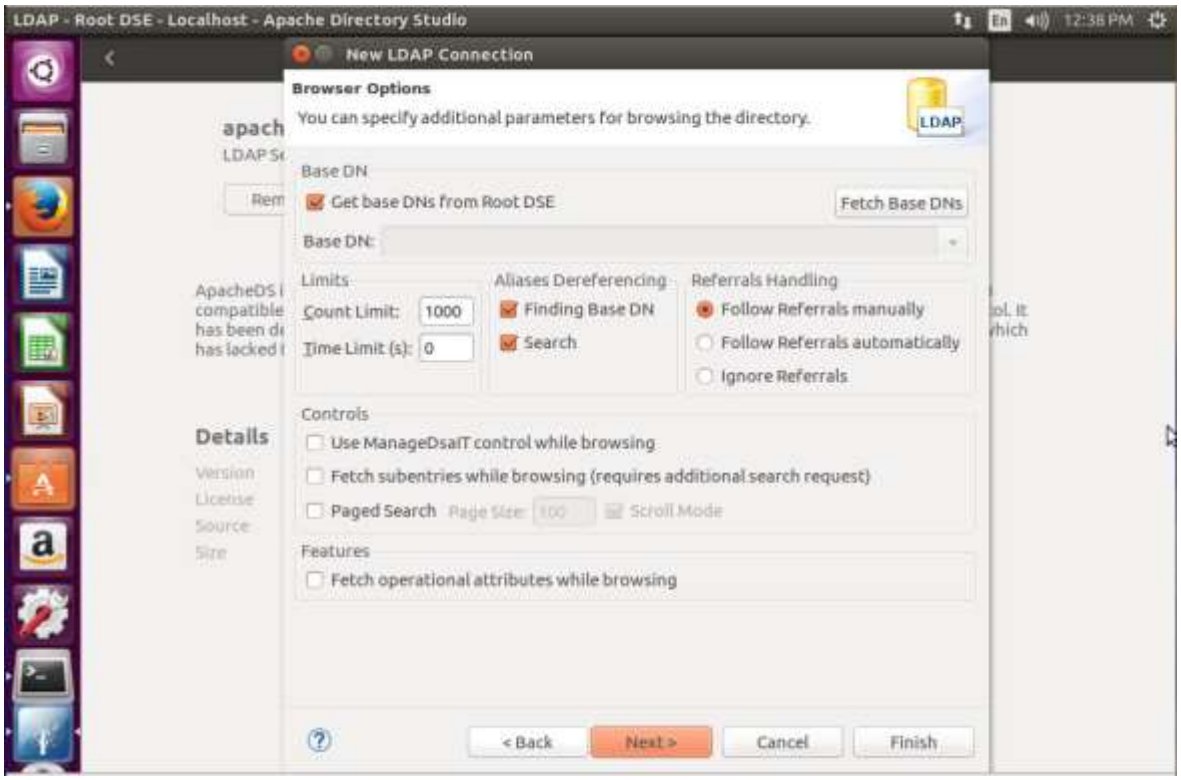

Figure 6: Additional options for your LDAP connection.

Finally you can specific parameters for editing entries on your LDAP server (Figure 7). Again, this will be determined by your needs and how you've setup your LDAP server.

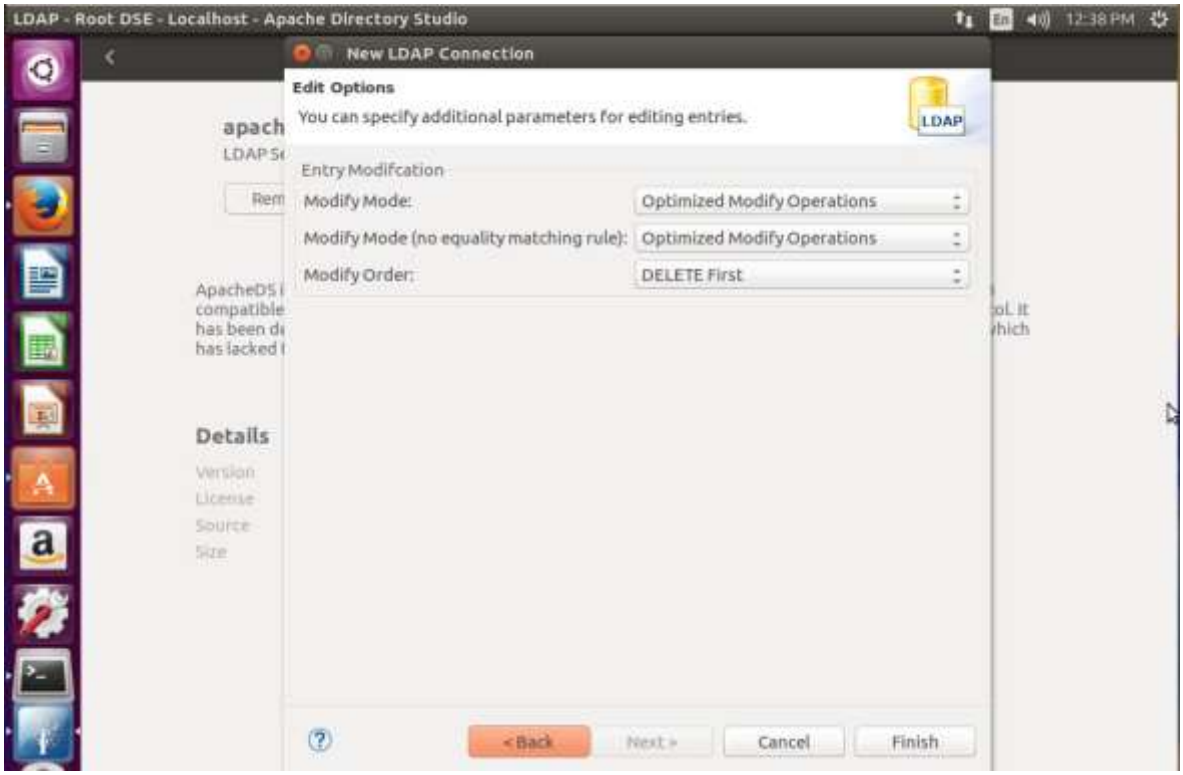

Figure 7: The final window of the connection wizard.

When the LDAP Browser window opens (Figure 8), you can then click on your *dc* entry and start working with LDAP.

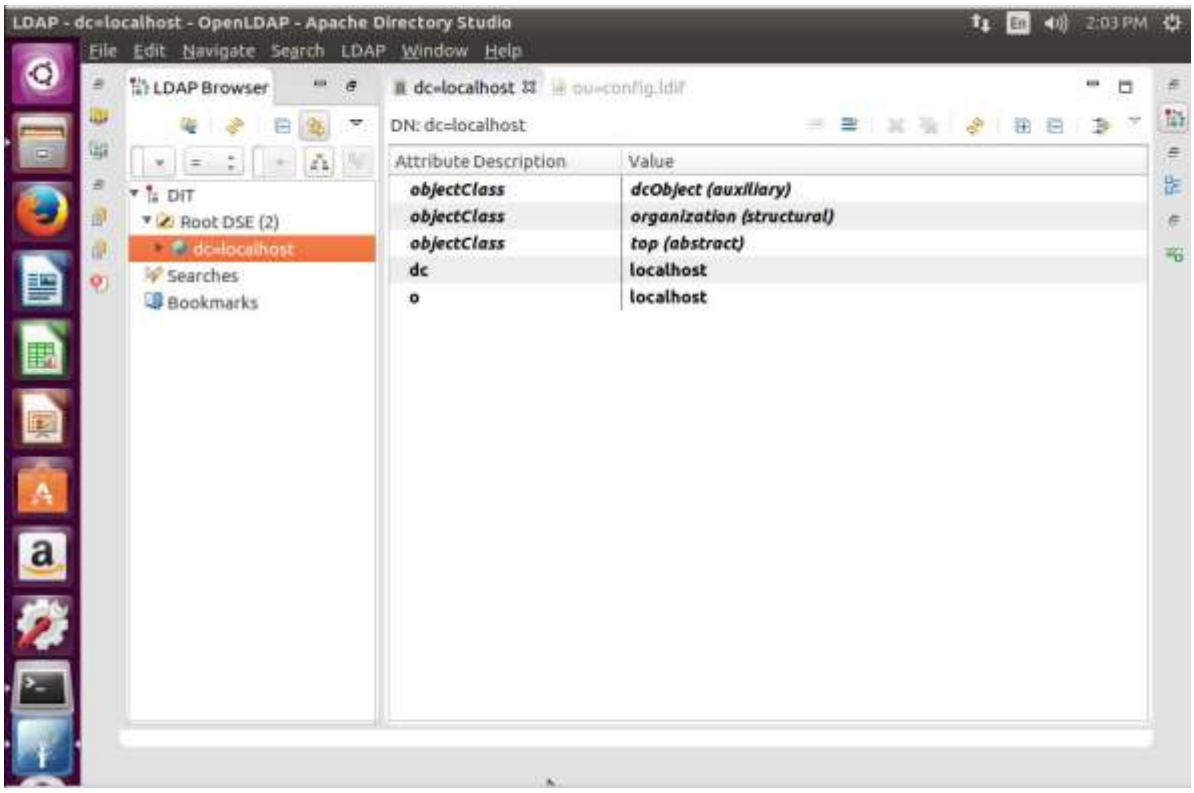

Figure 8: A successful LDAP connection.

To work with LDAP, you will right-click on the right pane and select the option you want to use (such as creating a new Attribute — Figure 9).

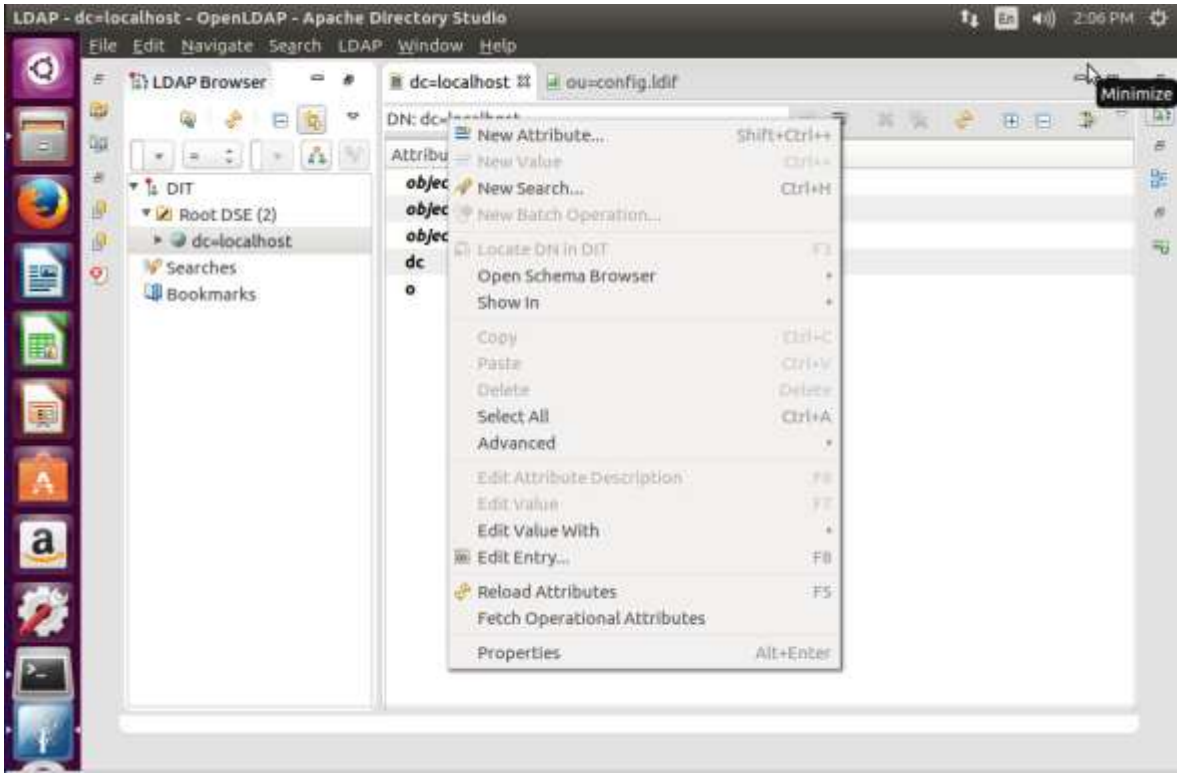

Figure 9: Creating a new Attribute in Apache Directory Studio.

Expand the *dc*= entry (in the left pane) and you can then start adding Users and Groups. Click on Users and then right-click *ou=Users*, select New, and you can then create from a long list of available object classes (Figure 10).

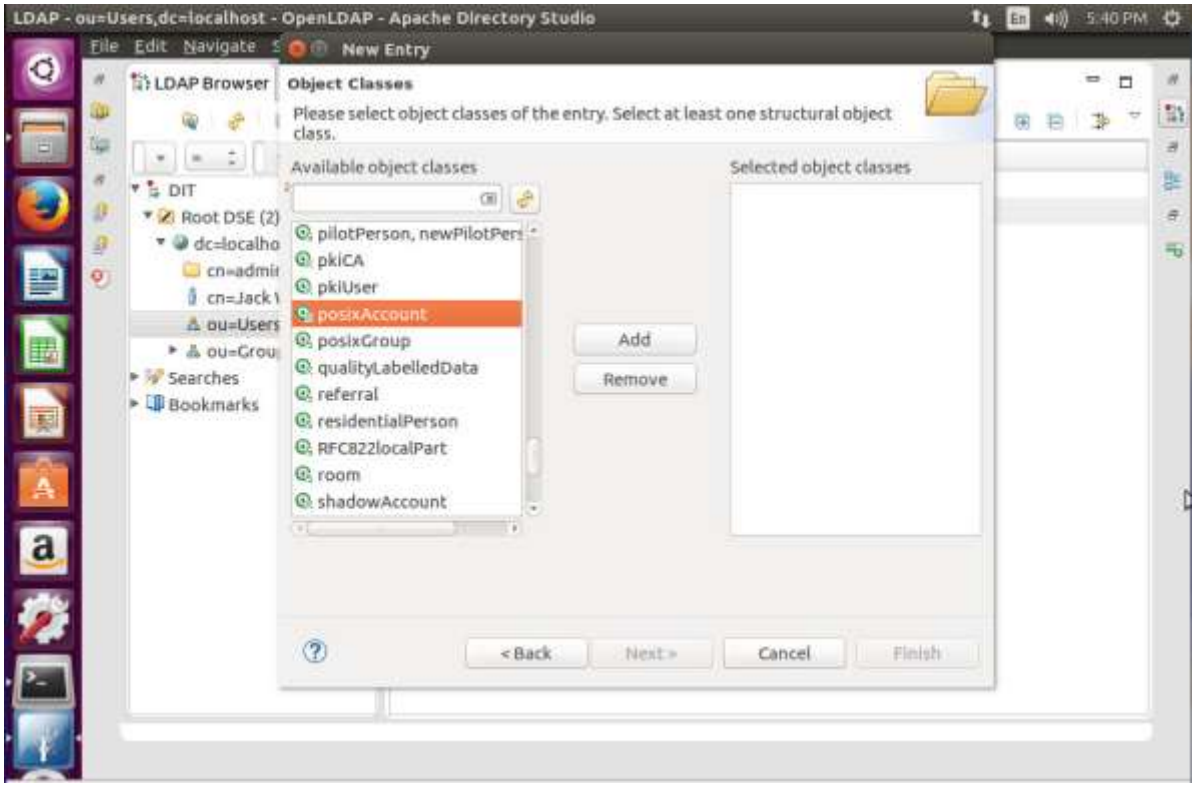

Figure 10: Creating a new object with Apache Directory Studio.

There you have it. You've successfully, installed, connected, and used the Apache Directory Studio to work with your existing LDAP server. You can now connect Apache Directory Studio to any of your LDAP servers and manage them all from a single point of entry.

Integrating LDAP With Tomcat

- ## Step 1: Get LDAP Server Information

  The first step in this process is to gather some information about your ldap server. If we don't have this information, then there isn't much further that we can go. To be able to successfully set up these permissions, we need to know the location of the ldap server, an ldap admin user id (on that preferably can only read the ldap contents, and not modify them), as well as the groups that the user should be a member of if they are granted access.

  For this example, we will rely on the following arbitrary information:

  - LDAP Server: ldap.normalexception.net
  - LDAP Port: 389
  - LDAP Admin: ldapreader

- LDAP Admin Membership: ou=Admins,o=normalexception.net
- LDAP Admin Pasword: readonly
- LDAP Userbase: ou=Members,o=normalexception.net

Step 2: Open Tomcat Server.xml File

The next step is to open the server configuration file for your tomcat installation. This is usually found in (TOMCAT_INSTALL_DIR)\conf\server.xml. You will want to scroll down to the server realm config section which should start off like:

<Realm className="org.apache.catalina.realm.UserDatabaseRealm" (in my case it was line 120).

This particular section tells tomcat that the authentication for the server is based off of the tomcat user configuration file (the tomcat-users.xml file). At this point in the tutorial, we will be replacing this with the ldap authentication realm. Later in this tutorial I will explain how to retain both an LDAP authentication as well as a user database authentication. What we want to do next, is to replace the<="" p="">

```
<Realm className="org.apache.catalina.realm.JNDIRealm"
 connectionName="uid=ldapreader,ou=Admins,o=normalexception.net"
 connectionPassword="readonly"
 connectionURL="ldap://ldap.normalexception.net:389"
 referrals="follow"
 userBase="ou=Members,o=normalexception.net"
 userSearch="(uid={0})"
 userSubtree="true"
 userRoleName="memberOf"
/>
```

There are a few things that the above will tell the server. The first is the class name of the realm object, which in the case of LDAP authentication will be "org.apache.catalina.realm.JNDIRealm". The second line 'connectionName' is the information about the ldap admin that will be reading the ldap database for authentication. As we mentioned in the previous step, this admin is 'ldapreader' and he belongs to the 'ou=Admins,o=normalexception.net' organizational unit. The 'connectionPassword' reflects the password of the admin, and the 'connectionURL' is the address to the ldap server.

'Referrals' specifies that you want the authenticator to do a referral search. In many cases you want this value to be 'follow' as many ID's and OU's have referral links. The 'userBase'

states what the unit is named where the members are all located, which in our case is 'ou=Members' which is found in 'o=normalexception.net'. The user search is simply '(uid={0})' which just tells the search that the value we are searching for is a 'uid'. You want to enable 'userSubtree' if you have a complex ldap tree structure. This ensures that you search the entire directory tree under the 'OU' to find the user. Note that this may increase search times on larger ldap servers. Lastly, you will want to find the users roles which in our case are found under the tag of 'memberOf'. This is default for LDAP.

• Step 3: Enabling LDAP Authentication In Your Webapp

Now that we enabled ldap authentication in our tomcat server, we need to tell our webapp that we want it to do an ldap authentication when a user accesses the page. To do this, we will navigate to our webapp's web.xml file which is generally found in (TOMCAT_INSTALL_DIR)\webapps\{webappname}\WEB-INF\web.xml.

When you open the web.xml file for your webapp, you will see a bunch of configuration parameters for your webapp. Chances are you will not have a section that is called '<security-constraint>', no worries, we are going to be adding that right now. Scroll to the end of your web.xml file and paste the following information:

```
<security-constraint>
 <web-resource-collection>
      <web-resource-name>LDAP Authentication</web-resource-name>
      <description>
      Authentication for registered users.
      </description>
      <url-pattern>/*</url-pattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
 </web-resource-collection>
 <auth-constraint>
      <role-name>regular_user</role-name>
 </auth-constraint>
</security-constraint>
<security-role>
 <role-name>regular_user</role-name>
</security-role>
<login-config>
 <auth-method>BASIC</auth-method>
 <realm-name>NormalException.net</realm-name>
</login-config>
```

The above block of text will tell our webapp that we will be using an LDAP security constraint to access the page. The properties are as follows:

- web-resource-name : The name of the resource, this is only used if we want to access the resource from within some java servlet code. Many users will not be doing this, so you can put whatever value you prefer here
- description : A simple description of what this authentication constraint is
- url-pattern : What url's do we want this security to apply to? In this case, we are making sure the user is authenticated for every page of the app
- http-method : What methods do we want authentication for? In my case I want authentication for GET's and POST's
- role-name : What role should the user belong to if they are granted access. In my sample ldap environment, all regular members are under the 'regular_user' role.
- auth-method : In most cases BASIC will be sufficient, but you can also have different authentication methods if you prefer more advanced authentication security. Please check the apache website for more information.
- realm-name : The name of our realm, this will be displayed in the pop-up dialog that requests the username and password from the user