Red Hat Cluster Suite and LDAP

V0.2 8 September 2009 Christine Caulfield <ccaulfie@redhat.com>

Starting with the 3.0.0 release of Red Hat cluster, pluggable configuration modules are supported. The main one people will use is 'xmlconfig' which reads the existing cluster.conf file directly from disk, but an LDAP one is also provided that allows the cluster configuration to be held in an LDAP database.

The tools for using LDAP to hold cluster information are best described as 'basic' at the moment, I hope this will improve (*ie* it really ought to be supported by the graphical front-end!)

Introduction to LDAP

I won't describe LDAP in much detail here, partly because I don't actually know much detail and partly because if you know nothing about LDAP then you probably need to stop reading this and go and find out about it before using it to configure something as complicated as a cluster.

LDAP is a directory service, it holds data centrally in its own database format and distributes it out to clients that know how to use the LDAP protocol. Objects in LDAP are roughly hierarchical, but when you connect to the server you can tell it at which point in the hierarchy you want to look, and this might be subject to access restrictions.

Many people seem to think that LDAP is a single point of failure in a cluster, but this isn't really the case. Most enterprise LDAP servers support redundancy and, even if you don't use it, it's not critical to the cluster that the LDAP server is up 99.9999% of the time. The cluster software only accesses the LDAP server when a node boots or the configuration is changed and reloaded. For the rest of the time the cluster nodes keep their own copy of the configuration data and use that for working with.

Having said that, I still recommend that production systems use a redundant LDAP server setup as you don't want to have to reboot a node only to find out that it can't read its configuration because the LDAP server has crashed!

Getting your configuration into LDAP

Sadly, this is by far the hardest bit, even if you are a hardened LDAP expert. Currently there are no user-friendly tools for doing it. You either have to hand-hack the LDAP database using an LDAP editor (if you want to do this I know a couple of good psychiatrists you might want to visit first), or create an LDIF file and load that into the LDAP server.

This is actually pretty rubbish. If you think about it, that makes making changes to the database somewhere between difficult and annoying. Yes, it is possible – but it's not something you want to be doing on a frequent basis.

The problem with editting the database online is that, because the cluster.conf structure is mirrored exactly in LDAP there are lots of empty 'structural' elements that also need creating, such as the <clusternodes> part.

The first thing you need to do, is to set up your LDAP server (which I am going to leave you to do unaided). I assume here that you are using the '389' LDAP server here (formerly known as Fedora Directory Server). I suspect that the procedure here will be roughly similar for other LDAP servers though.

Once you have a working LDAP server you need to load the schema for cluster suite into it. This is simply a matter of copying the file /usr/share/doc/cluster/99cluster.ldif file into the server's schema directory – usually /etc/dirsrv/slapd-<servername>/schema – and restarting the server.

Well, that was the easy bit. Now you have to get a cluster configuration from somewhere and load that into your shiny new (or perhaps well production-worn) LDAP server. Though actually I do recommend that you try this on a shiny new one first before you start making a mess of a production server's database.

Now, this might sound a little backwards, but I have generally found it easier to create the cluster.conf file using either ccs_tool/system-config-cluster/conga/$EDITOR_OF_YOUR CHOICE and then import that into LDAP. In the cluster tree there is a program called confdb2ldif which takes a running cluster configuration and turns it into an LDIF file that can be loaded into the LDAP server.

The syntax for running confdb2ldif is

```
# confdb2ldif <base DN>
```

where <base DN> is the base Distinguished Name (roughly: where it will sit in the LDAP hierarchy) for the cluster data. If you have several clusters you want to manage from the same LDAP server this will probably be at least 3 levels down. The example given in the usage text for the program gives

```
#confdb2ldif dc=mycompany,dc=com
```

which will be the toplevel DN for the server if your LDAP server node is host.mycompany.com. To make your cluster load from lower down in the hierarchy simple add the extra DN elements to the command-line eg:

```
# confdb2ldif cn=cluster1,dc=mycompany,dc=com
```

You will probably need to manually create any DN elements between dc=mycompany,dc=com and cn=cluster1. If you don't know what I mean when I talk about “base DNs” and “dc=” then you probably ought to review your LDAP skills.

Confdb2ldif writes its output to standard output so you will usually capture it to a file. If you're really confident then you could pipe it straight into the LDAP load tool. See later for the scary details.

By default confdb2ldif read from a running cluster. This might sound wrong, and it probably is. So there is a way to make it read a cluster.conf file that you have created by some means:

```
# COROSYNC_DEFAULT_CONFIG_IFACE=xmlconfig ./confdb2ldif <base DN>
```

or you can use a config file other than /etc/cluster/cluster.conf by putting that name into the COROSYNC_CLUSTER_CONFIG_FILE environment variable:

```
# COROSYNC_CLUSTER_CONFIG_FILE=/home/chrissie/cluster.conf -
COROSYNC_DEFAULT_CONFIG_IFACE=xmlconfig ./confdb2ldif <base DN>
```

If you have a strong constitution you might want to have a look at the generated file to see if it contains the things you expected it to contain. If it doesn't then check your cluster.conf file and/or file a bug.

Now you have an LDIF file containing the mangled remains of your cluster.conf file, it's time to load it into your LDAP server. To do this you will need the openldap-clients installed (that's the Fedora RPM package name, check your distribution for local differences).

```
# ldapadd -D 'cn=Directory Manager' -W -c < cluster.ldif
```

The password it asks you for is the password you gave when setting up the LDAP server. If you need to change any entries, you can regenerate the cluster.ldif file and reload it using ldapmodify with the same options above. You will need to ignore the duplicate entry errors in this case.

Actually, that's not really true. You can't actually *change* entries in this way, you can only add them (eg adding new nodes). To change entries ... well, you're own your own for the moment, sorry

If you've managed to follow all that, you can move onto the next step.

Configuring CMAN for LDAP

After all that you will be relieved to read that configuring cman to use LDAP rather than cluster.conf is considerably easier that getting the information into there.

The two things you need to tell cman are the base DN (remember that from the last section?) and the URL of the ldap server where the information is held, then you need to define three environment variables to tell the cman startup script (or cman_tool) to use it. LDAP URLs can get complicated but basically they are usually ldap://server with maybe an optional :port on the end.

COROSYNC_LDAP_URL

COROSYNC_LDAP_BASEDN

CONFIG_LOADER=ldapconfig

The first two should now be self-explanatory, CONFIG_LOADER is a variable that tells the cman init script which configuration plugin to use. This should be set to "ldapconfig". If you're not using the cman init script (and you probably should be unless you have a special reason not to), this is the argument passed to cman_tool via the -C switch. See the man page for cman_tool(8) for more information about plugins in general.

On a Red Hat (Fedora or RHEL) system these variables should go into /etc/sysconfig/cman.

Migrating from cluster.conf to LDAP

If you've really been paying attention, then the procedure for migrating a system from local cluster.conf files to LDAP should be fairly obvious by now, I hope: Simply take your existing cluster.conf file, feed it through confdb2ldif and load it into the server.

There is no need to take the cluster down to do this if you have not changed the file. And new/rebooted nodes can run from LDAP along with existing nodes that were booted from cluster.conf.

However, while running in this 'rolling upgrade' configuration you ***MUST NOT*** change the configuration and try to reload it, because some nodes will read the configuration from LDAP and some from cluster.conf. To make a node read it's configuration from LDAP means you need to change the environment variables listed above and reboot it (or, at least, restart the cluster on that node)

Updates

Much as some developers would like it to be, cluster configuration information isn't static, so there needs to be some way of updating it. Currently it's easiest to keep a cluster.conf file somewhere as a master file. Then, when things need to be changed you alter that file, generate a new LDIF file as above, and reload the configuration into LDAP. I know this is complicated, error-prone and annoying. Before reloading, it's best to remove the current configuration tree as follows:

```
# ldapdelete -D 'cn=Directory Manager' -W -r  -
'cn=cluster,dc=mycompany,dc=com'
```

and then load the new configuration as if it was a new one.

If you lose the cluster.conf file you were using as a 'master' then don't worry! You can generate a new one using ccs_config_dump command on a running cluster, or even from the current LDAP server:

```
COROSYNC_DEFAULT_CONFIG_IFACE=ldapconfig ccs_config_dump > cluster.conf
```

Limitations

Most of the limitations and annoyances have been mentioned in passing above, but I'll write them here so they don't get lost.

1. There are no sensible ways of creating/editting configurations directly in LDAP
2. Only the '389' LDAP server has been tested
3. You need to be confident with a Linux shell to do anything
4. During a rolling upgrade, the configuration must not be changed
5. confdb2ldif might not always be up-to-date

Example LDIF file