OpenLDAP in High Availability Environments

Ralf Haferkamp, rhafer@suse.com

September 2011

LDAP Directories are a central part in the IT infrastructure of a lot of organizations. Commonly many services in the organization rely on the availability of the Directory. Because of this unexpected outages or downtimes of the Directory Service can be very expensive and have to be kept to a minimum. There are several ways to increase the Availability of the Directory Service. This paper will highlight one of those ways by outlining a setup of software from various open source projects to create a high availability failover cluster of OpenLDAP servers.

1 Failover Clusters: A brief introduction

Failover clusters are central in most HA solutions. A failover setup consists of two or more redundant computers (also referred to as "nodes"). If one of the nodes fails, applications can be migrated ("failed over") to another node. In order to allow automatic failover a Cluster Management Software is running on the cluster. It has the following tasks to accomplish:

Membership: Establish which nodes are currently part (member) of the cluster. Make sure that only authorized nodes are able to become a member of the cluster.

Messaging: A working communication channel between all nodes is essential for the cluster. Through this channel the nodes can monitor each other and see if the cluster is still healthy. If the communication channel between one or more nodes of the cluster is lost the cluster becomes partitioned (also known as a *split-brain* condition), it is important that only one of the remaining partitions (sub-clusters) continues providing service. This can be determined through a vote of all remaining nodes. The partition that has more than half of the total possible votes will form a new cluster (it is said to have *quorum*). As the new formed cluster usually cannot determine if the partitioned nodes are dead or if it is just a communication problem between the nodes another important concept of HA clusters comes into place: Fencing.

Fencing: The goal of fencing is to avoid damage by preventing a (non-quorum) sub-cluster from accessing resources of the cluster. There are two different kinds of fencing:

- **Resource fencing:** Keep the fenced node from accessing the resources it might be trying to access. E.g. by reconfiguring switches to deny access for that specific node.
- **Node fencing:** Keep the fenced node from accessing any resource. Most easily implemented by resetting or powering off the node. This technique is also referred to as STONITH ("Shoot the other node in the Head")

Resources: Any type of application or service that is known to the cluster is called a resource. Resources can be a particular IP address, a database or an LDAP Server. The cluster management software is responsible for starting, stopping and monitoring the resources.

2 Failover Clusters on Linux

When setting up a Failover Cluster on Linux usually Pacemaker is used. Pacemaker provides an Open Source High Availability Cluster Resource Manager.

2.1 Pacemaker Architecture

From a very high level view a Pacemaker cluster is made up of these components:

Membership and Communication: Pacemaker can make use of different frameworks for communication and membership. This document only covers the Corosync Engine[1].

Cluster Resource Management Daemon (CRMd) `crmd` is the central daemon of the cluster coordinating all other subsystems.

Cluster Information Base (CIB): Represents (in XML) the configuration and the current state of all resources in the cluster. The CIB is automatically synchronized across all cluster members.

Policy Engine (PEngine): Uses the information from the CIB to make decision about resource management actions (e.g. which resources to start/stop on which nodes of the cluster).

Local Resource Manager (LRM): Responsible for managing the resources on a single node by invoking the respective resource agents.

Resource agents: They provide an abstract interface for the LRM to a specific resource (e.g. the OpenLDAP server) to give the cluster a consistent view on the resources. By that the cluster does not need to understand how exactly a resource works and can rely on the agent to do the right thing when it issues a `start`, `stop` or `monitor` action. Typically resource agents are written as normal shell scripts and are in many regards very similar to System V init scripts.

STONITHd: Pacemaker's fencing subsystem.

Management tools: Pacemaker come with a set of command line and GUI tools to manage and monitor the cluster and its resources.

2.2 Resources

As already mentioned all services/application that are managed by the cluster are called Resources. Pacemaker knows a few different types of resources:

Primitive: The most basic type of a resource, representing a single service.

[1] http://www.corosync.org

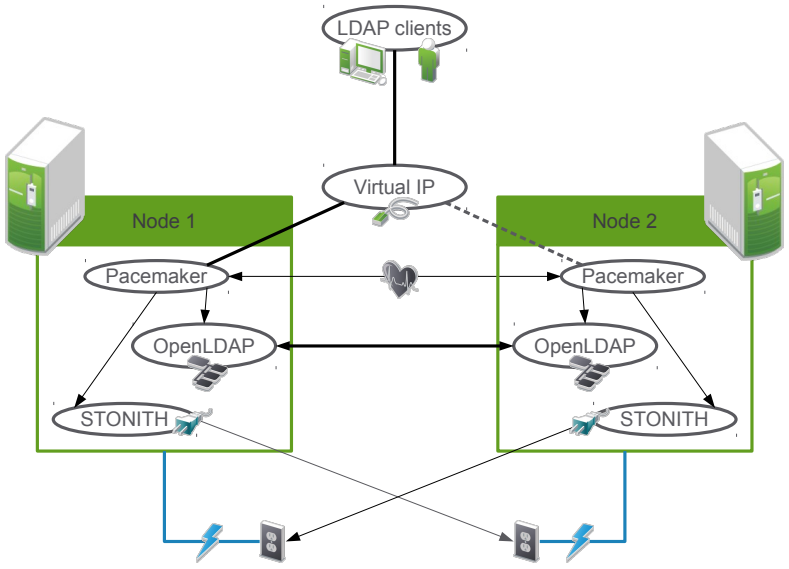
Group: A group is made of a set of primitive resources. Groups can be used to define certain dependencies that exists between the members of the group. E.g. the members need to be started in a specific order, or all group members need to run on the same node.

Clone: Clones are use to setup resources that need to run on multiple nodes at the same time. Clones can be created from a Group or a Primitive.

Master: A master resource is a specialization of a clone resource. It is used to represent resource that can operate in different modes (master or slave).

3 A Failover setup for OpenLDAP

The following sections introduce an example setup for a High Available OpenLDAP Service. It is a two-node setup where both nodes are running OpenLDAP using *MirrorMode* replication to keep the LDAP Data and Configuration synchronized. *MirrorMode* is a hybrid replication setup that provides all of the consistency guarantees of single-master replication, while also allowing to provide the high availability of multi-master[1]. For successful operation *MirrorMode* requires an external (to OpenLDAP) mechanism to ensure that LDAP write operations are only directed to one of the participating nodes. The setup introduced here uses a virtual IP address managed by Pacemaker to achieve this goal. The image below gives an overview about the components used in the setup. As illustrated there Pacemaker is also used to manage OpenLDAP (to start, stop and monitor) and to handle fencing (through STONITH).



3.1 Preparations

Network Configuration Each node in the cluster has two separate network interfaces. One is purely used for the cluster's internal communication (LDAP replication traffic and internal

communication of the different cluster components). The other one is used for the external traffic, i.e. mainly the LDAP clients accessing the LDAP service. In this setup the cluster internal interfaces will use IP Addresses from the `192.168.100.0/24` subnet while the external/public interfaces are on the `192.168.101.0/24` network.
It is important that every node is able to resolve the hostnames of all participating nodes to the correct IP Addresses. Either by setting up DNS or by adding the required entries to the /etc/hosts file.
Communication is essential to clusters. It is highly recommended that the communication paths between nodes are redundant, either by bonding the interfaces or by deploying the corosync Redundant Ring Protocol (RRP).

Time Synchronization For MirrorMode Replication to work accurately it is important that the system clocks on the participating nodes are tightly in sync. So it is recommended to setup NTP to synchronize the clocks of all nodes.

Certificates To have TLS encryption work as seamless as possible for the clients, special care has to be taken when creating the server certificates. The easiest setup probably is to create a single certificate for the whole cluster that is copied to both nodes. That certificate would need to have the names (and/or IP Addresses) of the internal and external interfaces of all participating nodes present in its `Subject Alt Name` Attribute, including the Virtual IP-Address and Name under which the server will be contacted by the clients.
Before describing the details of the Pacemaker setup we will now continue to layout the steps required to create an OpenLDAP MirrorMode configuration.

3.2 OpenLDAP MirrorMode Configuration

As OpenLDAP is able to replicate its own configuration database (`cn=config`) via SyncRepl the example setup will make use of this feature. Even though it is not strictly required for a MirrorMode setup, it will make configuration changes in the running cluster easier. The LDIF examples following in the next sections assume that slapd is already running on one of the nodes in a basic setup with a single `hdb` database serving the `dc=example,dc=com` subtree.

Syncrepl Account As the first step we create a new account object that is later used for authenticating the syncrepl consumer connections between the two nodes.

```
dn: uid=syncrepl,dc=example,dc=com
objectclass: account
objectclass: simpleSecurityObject
userPassword: {SSHA}ib8NBYz/pJVrpm/KKtjOQEJFzYVAPbxE
```

This account needs to have read access to every LDAP entry that is going to be replicated. So ACLs and Limits might need to be adjusted accordingly:

```
dn: olcDatabase={0}config,cn=config
add: olcLimits
olcLimits: dn.exact="uid=syncrepl,dc=example,dc=com"
  size=unlimited
```

```
-
add: olcAccess
olcAccess: {0}to *
  by dn.exact="uid=syncrepl,dc=example,dc=com" read
  by * break
```

Similar changes are needed for the `olcDatabase={1}hdb,cn=config` Database.

Server ID One major difference between the nodes in a MirrorMode setup and a normal syncrepl provider in master-slave Replication is the definition of the `olcServerId` attribute. Every node needs to have a unique server ID mapped to its hostname. The following LDIF is used with `ldapmodify` to add the Server IDs for the two node setup.

```
dn: cn=config
add: olcServerId
olcServerId: 1 ldap://<node1>
olcServerId: 2 ldap://<node2>
```

LDAPSync Replication Provider Every replicated database needs to have the SyncRepl provider overlay (`syncprov`) loaded. In LDIF for usage with `ldapadd`:

```
dn: olcOverlay=syncprov,olcDatabase={0}config,cn=config
objectclass: olcSyncprovConfig
olcSpCheckpoint: 100 5

dn: olcOverlay=syncprov,olcDatabase={1}hdb,cn=config
objectclass: olcSyncprovConfig
olcSpCheckpoint: 100 5
```

LDAPSync Replication Consumer The next step is to add the consumer configurations to the database. Note that each database will have multiple `olcSyncrepl` attributes pointing to all MirrorMode Nodes. Including one `olcSyncrepl` value pointing to itself. The SyncRepl client in `slapd` is smart enough to just ignore that one.

```
dn: olcDatabase={0}config,cn=config
add: olcSyncrepl
olcSyncrepl: rid=001
  provider="ldap://<node1>"
  searchbase="cn=config"
  type=refreshAndPersist
  bindmethod=simple
  binddn="uid=syncrepl,dc=example,dc=com"
  credentials="secret"
  retry="30 +"
```

```
  network-timeout=5
  timeout=30
  starttls="critical"
  tls_reqcert="demand"
olcSyncrepl: rid=002
  provider="ldap://<node2>"
  searchbase="cn=config"
  type=refreshAndPersist
  bindmethod=simple
  binddn="uid=syncrepl,dc=example,dc=com"
  credentials="secret"
  retry="30 +"
  network-timeout=5
  timeout=30
  starttls="critical"
  tls_reqcert="demand"
-
add: olcMirrorMode
olcMirrorMode: TRUE
```

The SyncRepl configuration for the `olcDatabase={1}hdb,cn=config` database is added in a similar way. The value for `rid` has to be unique.
It is important to add the `olcSyncrepl` and `olcMirrorMode` values in single Modification operation at least to the `olcDatabase={0}config,cn=config` database. Otherwise `cn=config` would become a read-only/slave database and would not accept configuration changes anymore.

3.3 Second Node

Once the configuration on the first node is completed, the second node can be initialized by dumping the first node's configuration to LDIF with:

```
slapcat -bcn=config -F/etc/openldap/slapd.d/ -l config.ldif
```

And reloading it on the second node with:

```
slapadd -bcn=config -F/etc/openldap/slapd.d/ -l config.ldif
```

Make sure that the Server Certificate and CA files are installed in the correct place on both nodes and that the user running slapd has sufficient access to them. After that `slapd` can be started on the second node. It should automatically start replicating the `hdb` database from the first node. The basic MirrorMode setup is now completed.
The remaining sections will describe how that setup is now integrated into a Pacemaker cluster. Before continuing it is important to stop slapd and to disable any init-scripts that would start it again automatically at boot-time. Starting and stopping slapd will later be handled by the cluster stack.

4 Pacemaker/Corosync Setup

4.1 Required Software

At least the following software components need to be installed for the example setup:

- PaceMaker
- Resource Agents
- CoroSync
- ClusterGlue

SUSE offers the *High Availability Extensions* Addon for *SUSE Linux Enterprise Server 11* [2]. All of the above mentioned components are part of that product and the configuration examples throughout the following section are based on that product. Prebuilt packages of the required components are also available for various other Distributions [3].

4.2 Basic Setup

The main configuration is stored in `/etc/corosync/corosync.conf`. It is divided into multiple sections. The `totem` section contains setup for the communication channels used by the cluster nodes. Each communication channel is defined in a separate `interface` subsection inside that section. As Corosync uses multicast for communication a port number and multicast address for use by the cluster is defined. The example below uses `226.94.1.1` as the multicast address and the port number `5405`. Additionally the network address of the interface to use for cluster communication needs to be defined (here: `192.168.100.0`).

```
totem {
        version:        2
        token:          5000
        token_retransmits_before_loss_const: 10
        join:           60
        consensus:      6000
        vsftype:        none
        max_messages:   20
        clear_node_high_bit: yes
        secauth:        on
        threads:        1
        interface {
                ringnumber: 0
                bindnetaddr: 192.168.100.0
                mcastaddr: 226.94.1.1
                mcastport: 5405
                ttl: 1
        }
}
```

[2] `http://www.suse.com/products/highavailability/`
[3] `http://www.clusterlabs.org/wiki/Install`

The `service` Section is used to enable the Pacemaker CRM and the management daemon which is used by the various management tools to access the cluster configuration:

```
service {
        ver:            0
        name:           pacemaker
        use_mgmtd:      yes
        use_logd:       yes
}
```

In the `aisexec` section we configure Corosync to run as the `root` user:

```
aisexec {
        user: root
        group: root
}
```

For encrypted cluster communication a shared key needs to be generated. The `corosync-keygen` command is used for that. The key will end up in `/etc/corosync/authkey`. This file together with `corosync.conf` should be copied over to the second cluster node. (Hint: In larger clusters a tool like `csync2`[4] can be used for distributing the configuration.)

4.2.1 Starting and Verifying the Configuration

The cluster software can now be started to verify the configuration. On SUSE Linux Enterprise it is done by running this command on every node:

```
/etc/init.d/openais start
```

To check the cluster status now the `crm_mon` command is used. If all nodes are online the output should be similar to this:

```
============
Last updated: Mon Sep 26 13:50:22 2011
Last change: Mon Sep 26 12:10:52 2011 by hacluster via crmd on node1
Stack: openais
Current DC: node1 - partition with quorum
Version: 1.1.6-9971ebba4494012a93c03b40a2c58ec0eb60f50c
2 Nodes configured, 2 expected votes
0 Resources configured.
============

Online: [ node1 node2 ]
```

After these basic configuration steps are done we can now start to setup the cluster resources.

[4] `http://oss.linbit.com/csync2/`

4.3 Resources

As already mentioned the cluster configuration is done either via the command line interface `crm` or the GUI interface `crm_gui`. This paper will focus on the command line tool. The tool offers a set of subcommands to manage the cluster configuration. Use `crm help` to get an overview of all the supported subcommands. The `crm` command can also be used as an interactive shell including tab completion and command history. For that just execute `crm` without any arguments. To leave the shell use the commands `quit`, `bye` or `exit`.
The first step is now to define what to do when the cluster has no quorum. As we are setting up a two node cluster, quorum is always lost when a single node is failing. So we configure the cluster to continue "normal" operations when quorum is lost. This is done by setting the global property `no-quorum-policy` to `ignore`.

```
node1:~ # crm configure property no-quorum-policy=ignore
```

The `crm` tool offers the `ra` subcommand to get information about the available resource agents. Pacemaker supports different classes of resource agents implementing different standard interfaces. To get a list of the resource agent classes use the command:

```
node1:~ # crm ra classes
heartbeat
lsb
ocf / heartbeat pacemaker
stonith
```

To get a list of all available resource agents for a class use the list command:

```
node1:~ # crm ra list ocf
AoEtarget           AudibleAlarm        CTDB
[..]
tomcat              vmware
```

To display detailed documentation about a resource agent and required parameters use the info subcommand:

```
node1:~ # crm ra info ocf:heartbeat:IPaddr2
```

4.3.1 Fencing

In Pacemaker node level fencing is implemented with the agents of the `stonith` class. As the example setup is running on virtual machines the `external/libvirt` STONITH agent is used. In a production setup on real hardware usually some kind of remote power switch is used for the STONITH resource.
The `external/libvirt` agent accesses the virtualization host's management daemon `libvirtd` to reset the node that is supposed to be fenced. It requires the following parameters:

hostlist: A list of `hostname:domain_id` tuples of the controlled nodes. The `domain_id` can be omitted if it matches the hostname.

hypervisor_uri: URI for connection to libvirtd. E.g. `qemu+ssh://kvm.site/system` for access via ssh.

To use this agent `virsh` must be installed on the node (part of the libvirt-client package on SLES) and access control on the hypervisor must be configured to give sufficient privileges for the selected URI. To add the STONITH configuration as a cloned resource, so that it is active on every node, use:

```
node1:~ # crm configure
crm(live)configure# primitive st0 stonith:external/libvirt \
              params hostlist="node1:node1 node2:node2" \
              hypervisor_uri="qemu+ssh://192.168.101.1/system" \
              op monitor interval=15s
crm(live)configure# clone stonith st0
crm(live)configure# commit
```

This also configures the external/libvirt agent's monitor operation to check connectivity to the hypervisor every 15 seconds.

4.3.2 OpenLDAP Resource Agent

Up to now there is no working pre-packaged resource agent for OpenLDAP available. There is however one in review on the linux-ha developers mailing list currently[5] which works reasonably well already. To be used it needs to be installed in the `/usr/lib/ocf/resource.d/heartbeat/` directory. Use `crm ra info ocf:heartbeat:slapd` to display detailed information about all it supported parameters. To configure with the `crm` interactive shell use:

```
node1:~ # crm configure
crm(live)configure# primitive slapd_mirrormode ocf:heartbeat:slapd params\
                        slapd="/usr/lib/openldap/slapd" \
                        config="/etc/openldap/slapd.d/" \
                        user="ldap" group="ldap" \
                        services="ldap:/// ldaps:/// ldapi:///" \
                        watch_suffix="dc=example,dc=com" \
                        meta migration-threshold="3" \
                        op monitor interval=10s
```

This sets up a primitive resource named `slapd_mirrormode` the using the parameters:

slapd: Contains the path to the `slapd` binary

config: Contains the path to either the `slapd.conf` configuration file or the directory which is used of the `cn=config` database.

[5] `https://github.com/jhohm/resource-agents/blob/master/heartbeat/slapd`

user/group: User/Group which slapd should run as. (Passed to slapd's `-u -g` arguments.

services: Defines which interfaces slapd should listen on (Passed to slapd's `-h` argument.

watch_suffix: The suffix of the database(s) which the agent should monitor for availability. (Multiple suffixes are possible, it is also possible to configure bind-credentials to use an authenticated connection for for monitoring, see the agent's documentation for details)

The meta Attribute `migration-threshold` is set to `3` for this resource. This means that pacemaker will try to start/restart the resource 3 times before it stops trying to run the resource on a specific node.
The monitor operation of the slapd agent is configured to run every 10 seconds.
Now we tell crm to setup the `slapd_mirrormode` primitive as a cloned resource to have it running on all nodes and activate the configuration using the `commit` command.

```
crm(live)configure# clone ldap_clone slapd_mirrormode
crm(live)configure# commit
```

If everything went right `slapd` should be running on both nodes in the cluster now. Use `crm_mon` to verify that. Its output should contain something like:

```
Clone Set: ldap_clone [slapd_mirrormode]
  Started: [ node2 node1 ]
```

4.3.3 Virtual IP

The `IPaddr2` resource agent is used to set the Virtual IP Address. The monitoring interval is set to 5 seconds:

```
crm(live)configure# primitive ldap-v_ip ocf:heartbeat:IPaddr2 \
        params ip="192.168.101.200" nic="eth0" \
        op monitor interval="5s" timeout="20s"
```

We also need to define that the new `ldap-v_ip` resource needs to run together with one of the cloned slapd instances. This is done with a colocation constraint:

```
crm(live)configure# colocation v_ip_with_slapd inf: ldap-v_ip ldap_clone
```

After that an order constraint is added to make sure that the Virtual IP resource is always started before the `ldap_clone` resource. Again the `commit` command is needed to activate the configuration.

```
crm(live)configure# order ip_before_slapd inf: ldap-v_ip ldap_clone
crm(live)configure# commit
```

4.4 Testing and Monitoring

The setup is now basically complete and ready for some testing. Monitoring the current status of the cluster can be done with the `crm_mon` tool. For testing if the failover of the virtual IP address correctly works when slapd crashes on the currently active node, you can for example kill the slapd process on that node. Pacemaker will first try to restart slapd. After slapd was killed for the third time, pacemaker should re-assign the virtual IP address to the second node. After that the `slapd_mirrormode` resource should be shown as `stopped` on the failed node. Further testing includes checking whether fencing is working correctly when e.g. the communication channel between the cluster nodes is lost.

5 Possibilities for enhancing the Setup

The setup introduced here is a very simple example of an HA cluster for OpenLDAP. There are various options to enhance this. E.g. some additional OpenLDAP nodes could be added acting as SyncRepl consumers of the MirrorMode nodes. LDAP traffic could be load-balanced among these nodes leveraging e.g. Linux Virtual Server (LVS). The SyncRepl consumer would be configured to chain LDAP write operation to the virtual IP configured for the MirrorMode nodes.

References

[1] The OpenLDAP Project, *OpenLDAP Software 2.4 Administrator's Guide*, `http://www.openldap.org/doc/admin24/index.html` 2011

[2] Beekhof, Andrew, *Pacemaker 1.1 Configuration Explained*, `http://www.clusterlabs.org/doc/en-US/Pacemaker/1.1/pdf/Pacemaker_Explained/Pacemaker-1.1-Pacemaker_Explained-en-US.pdf`, 2009

[3] Muhamedagic, Dejan, *The road to high availability: solutions built on open source*, `http://www.linux-ha.org/wiki/File:Linuxtag-09-ha-paper.pdf`, 2006

[4] Novell Inc., *SUSE Linux Enterprise High Availability Extension, High Availability Guide*, `http://www.suse.com/documentation/sle_ha/`, 2011