Contents | Parent Topic | Previous Topic | Next Topic
Home | Catalog

18. Replication

Replicated directories are a fundamental requirement for delivering a resilient enterprise deployment.

OpenLDAP has various configuration options for creating a replicated directory. In previous releases, replication was discussed in terms of a *master* server and some number of *slave* servers. A master accepted directory updates from other clients, and a slave only accepted updates from a (single) master. The replication structure was rigidly defined and any particular database could only fulfill a single role, either master or slave.

As OpenLDAP now supports a wide variety of replication topologies, these terms have been deprecated in favor of *provider* and *consumer*: A provider replicates directory updates to consumers; consumers receive replication updates from providers. Unlike the rigidly defined master/slave relationships, provider/consumer roles are quite fluid: replication updates received in a consumer can be further propagated by that consumer to other servers, so a consumer can also act simultaneously as a provider. Also, a consumer need not be an actual LDAP server; it may be just an LDAP client.

The following sections will describe the replication technology and discuss the various replication options that are available.

18.1. Replication Technology

18.1.1. LDAP Sync Replication

The LDAP Sync Replication engine, syncrepl for short, is a consumer-side replication engine that enables the consumer LDAP server to maintain a shadow copy of a DIT fragment. A syncrepl engine resides at the consumer and executes as one of the *slapd*(8) threads. It creates and maintains a consumer replica by connecting to the replication provider to perform the initial DIT content load followed either by periodic content polling or by timely updates upon content changes.

Syncrepl uses the LDAP Content Synchronization protocol (or LDAP Sync for short) as the replica synchronization protocol. LDAP Sync provides a stateful replication which supports both pull-based and push-based synchronization and does not mandate the use of a history store. In pull-based replication the consumer periodically polls the provider for updates. In push-based replication the consumer listens for updates that are sent by the provider in realtime. Since the protocol does not require a history store, the provider does not need to maintain any log of updates it has received (Note that the syncrepl engine is extensible and additional replication protocols may be supported in the future.).

Syncrepl keeps track of the status of the replication content by maintaining and exchanging synchronization cookies. Because the syncrepl consumer and provider maintain their content status, the consumer can poll the provider content to perform incremental synchronization by asking for the entries required to make the consumer replica up-to-date with the provider content. Syncrepl also enables convenient management of replicas by maintaining replica status. The consumer replica can be constructed from a consumer-side or a provider-side backup at any synchronization status. Syncrepl can automatically resynchronize the consumer replica up-to-date with the current provider content.

Syncrepl supports both pull-based and push-based synchronization. In its basic refreshOnly synchronization mode, the provider uses pull-based synchronization where the consumer servers need not be tracked and no history information is maintained. The information required for the provider to process periodic polling requests is contained in the synchronization cookie of the request itself. To optimize the pull-based synchronization, syncrepl utilizes the present phase of the LDAP Sync protocol as well as its delete phase, instead of falling back on frequent full reloads. To further optimize the pull-based synchronization, the provider can maintain a per-scope session log as a history store. In its refreshAndPersist mode of synchronization, the provider uses a push-based synchronization. The provider keeps track of the consumer servers that have requested a persistent search and sends them necessary updates as the provider replication content gets modified.

With syncrepl, a consumer server can create a replica without changing the provider's configurations and without restarting the provider server, if the consumer server has appropriate access privileges for the DIT fragment to be replicated. The consumer server can stop the replication also without the need for provider-side changes and restart.

Syncrepl supports partial, sparse, and fractional replications. The shadow DIT fragment is defined by a general search criteria consisting of base, scope, filter, and attribute list. The replica content is also subject to the access privileges of the bind identity of the syncrepl replication connection.

18.1.1.1. The LDAP Content Synchronization Protocol

The LDAP Sync protocol allows a client to maintain a synchronized copy of a DIT fragment. The LDAP Sync operation is defined as a set of controls and other protocol elements which extend the LDAP search operation. This section introduces the LDAP Content Sync protocol only briefly. For more information, refer to RFC4533.

The LDAP Sync protocol supports both polling and listening for changes by defining two respective synchronization operations: *refreshOnly* and *refreshAndPersist*. Polling is implemented by the *refreshOnly* operation. The consumer polls the provider using an LDAP Search request with an LDAP Sync control attached. The consumer copy is synchronized to the provider copy at the time of polling using the information returned in the search. The provider finishes the search operation by returning *SearchResultDone* at the end of the search operation as in the normal search. Listening is implemented by the *refreshAndPersist* operation. As the name implies, it begins with a search, like refreshOnly. Instead of finishing the search after returning all entries currently matching the search criteria, the synchronization search remains persistent in the provider. Subsequent updates to the synchronization content in the provider cause additional entry updates to be sent to the consumer.

The *refreshOnly* operation and the refresh stage of the *refreshAndPersist* operation can be performed with a present phase or a delete phase.

In the present phase, the provider sends the consumer the entries updated within the search scope since the last synchronization. The provider sends all requested attributes, be they changed or not, of the updated entries. For each unchanged entry which remains in the scope, the provider sends a present message consisting only of the name of the entry and the synchronization control representing state present. The present message does not contain any attributes of the entry. After the consumer receives all update and present entries, it can reliably determine the new consumer copy by adding the entries added to the provider, by replacing the entries modified at the provider, and by deleting entries in the consumer copy which have not been updated nor specified as being present at the provider.

The transmission of the updated entries in the delete phase is the same as in the present phase. The provider sends all the requested attributes of the entries updated within the search scope since the last synchronization to the consumer. In the delete phase, however, the provider sends a delete message for each entry deleted from the search scope, instead of sending present messages. The delete message consists only of the name of the entry and the synchronization control representing state delete. The new consumer copy can be determined by adding, modifying, and removing entries according to the synchronization control attached to the *SearchResultEntry* message.

In the case that the LDAP Sync provider maintains a history store and can determine which entries are scoped out of the consumer copy since the last synchronization time, the provider can use the delete phase. If the provider does not maintain any history store, cannot determine the scoped-out entries from the history store, or the history store does not cover the outdated synchronization state of the consumer, the provider should use the present phase. The use of the present phase is much more efficient than a full content reload in terms of the synchronization traffic. To reduce the synchronization traffic further, the LDAP Sync protocol also provides several optimizations such as the transmission of the normalized `entryUUID`s and the transmission of multiple `entryUUID`s in a single *syncIdSet* message.

At the end of the *refreshOnly* synchronization, the provider sends a synchronization cookie to the consumer as a state indicator of the consumer copy after the synchronization is completed. The consumer will present the received cookie when it requests the next incremental synchronization to the provider.

When *refreshAndPersist* synchronization is used, the provider sends a synchronization cookie at the end of the refresh stage by sending a Sync Info message with refreshDone=TRUE. It also sends a synchronization cookie by attaching it to *SearchResultEntry* messages generated in the persist stage of the synchronization search. During the persist stage, the provider can also send a Sync Info message containing the synchronization cookie at any time the provider wants to update the consumer-side state indicator.

In the LDAP Sync protocol, entries are uniquely identified by the `entryUUID` attribute value. It can function as a reliable identifier of the entry. The DN of the entry, on the other hand, can be changed over time and hence cannot be considered as the reliable identifier. The `entryUUID` is attached to each *SearchResultEntry* or *SearchResultReference* as a part of the synchronization control.

18.1.1.2. Syncrepl Details

The syncrepl engine utilizes both the *refreshOnly* and the *refreshAndPersist* operations of the LDAP Sync protocol. If a syncrepl specification is included in a database definition, *slapd*(8) launches a syncrepl engine as a *slapd*(8) thread and schedules its execution. If the *refreshOnly* operation is specified, the syncrepl engine will be rescheduled at the interval time after a synchronization operation is completed. If the *refreshAndPersist* operation is specified, the engine will remain active and process the persistent synchronization messages from the provider.

The syncrepl engine utilizes both the present phase and the delete phase of the refresh synchronization. It is possible to configure a session log in the provider which stores the `entryUUID`s of a finite number of entries deleted from a database. Multiple replicas share the same session log. The syncrepl engine uses the delete phase if the session log is present and the state of the consumer server is recent enough that no session log entries are truncated after the last synchronization of the client. The syncrepl engine uses the present phase if no session log is configured for the replication content or if the consumer replica is too outdated to be covered by the session log. The current design of the session log store is memory based, so the information contained in the session log is not persistent over multiple provider invocations. It is not currently supported to access the session log store by using LDAP operations. It is also not currently supported to impose access control to the session log.

As a further optimization, even in the case the synchronization search is not associated with any session log, no entries will be transmitted to the consumer server when there has been no update in the replication context.

The syncrepl engine, which is a consumer-side replication engine, can work with any backends. The LDAP Sync provider can be configured as an overlay on any backend, but works best with the *back-bdb back-hdb*, or *back-mdb* backends.

The LDAP Sync provider maintains a `contextCSN` for each database as the current synchronization state indicator of the provider content. It is the largest `entryCSN` in the provider context such that no transactions for an entry having smaller `entryCSN` value remains outstanding. The `contextCSN` could not just be set to the largest issued `entryCSN` because `entryCSN` is obtained before a transaction starts and transactions are not committed in the issue order.

The provider stores the `contextCSN` of a context in the `contextCSN` attribute of the context suffix entry. The attribute is not written to the database after every update operation though; instead it is maintained primarily in memory. At database start time the provider reads the last saved `contextCSN` into memory and uses the in-memory copy exclusively thereafter. By default, changes to the `contextCSN` as a result of database updates will not be written to the database until the server is cleanly shut down. A checkpoint facility exists to cause the `contextCSN` to be written out more frequently if desired.

Note that at startup time, if the provider is unable to read a `contextCSN` from the suffix entry, it will scan the entire database to determine the value, and this scan may take quite a long time on a large database. When a `contextCSN` value is read, the database will still be scanned for any `entryCSN` values greater than it, to make sure the `contextCSN` value truly reflects the greatest committed `entryCSN` in the database. On databases which support inequality indexing, setting an eq index on the `entryCSN` attribute and configuring *contextCSN* checkpoints will greatly speed up this scanning step.

If no `contextCSN` can be determined by reading and scanning the database, a new value will be generated. Also, if scanning the database yielded a greater `entryCSN` than was previously recorded in the suffix entry's `contextCSN` attribute, a checkpoint will be immediately written with the new value.

The consumer also stores its replica state, which is the provider's `contextCSN` received as a synchronization cookie, in the

`contextCSN` attribute of the suffix entry. The replica state maintained by a consumer server is used as the synchronization state indicator when it performs subsequent incremental synchronization with the provider server. It is also used as a provider-side synchronization state indicator when it functions as a secondary provider server in a cascading replication configuration. Since the consumer and provider state information are maintained in the same location within their respective databases, any consumer can be promoted to a provider (and vice versa) without any special actions.

Because a general search filter can be used in the syncrepl specification, some entries in the context may be omitted from the synchronization content. The syncrepl engine creates a glue entry to fill in the holes in the replica context if any part of the replica content is subordinate to the holes. The glue entries will not be returned in the search result unless *ManageDsaIT* control is provided.

Also as a consequence of the search filter used in the syncrepl specification, it is possible for a modification to remove an entry from the replication scope even though the entry has not been deleted on the provider. Logically the entry must be deleted on the consumer but in *refreshOnly* mode the provider cannot detect and propagate this change without the use of the session log on the provider.

For configuration, please see the Syncrepl section.

18.2. Deployment Alternatives

While the LDAP Sync specification only defines a narrow scope for replication, the OpenLDAP implementation is extremely flexible and supports a variety of operating modes to handle other scenarios not explicitly addressed in the spec.

18.2.1. Delta-syncrepl replication

- Disadvantages of LDAP Sync replication:

LDAP Sync replication is an object-based replication mechanism. When any attribute value in a replicated object is changed on the provider, each consumer fetches and processes the complete changed object, including **both the changed and unchanged attribute values** during replication. One advantage of this approach is that when multiple changes occur to a single object, the precise sequence of those changes need not be preserved; only the final state of the entry is significant. But this approach may have drawbacks when the usage pattern involves single changes to multiple objects.

For example, suppose you have a database consisting of 102,400 objects of 1 KB each. Further, suppose you routinely run a batch job to change the value of a single two-byte attribute value that appears in each of the 102,400 objects on the master. Not counting LDAP and TCP/IP protocol overhead, each time you run this job each consumer will transfer and process **100 MB** of data to process **200KB of changes!**

99.98% of the data that is transmitted and processed in a case like this will be redundant, since it represents values that did not change. This is a waste of valuable transmission and processing bandwidth and can cause an unacceptable replication backlog to develop. While this situation is extreme, it serves to demonstrate a very real problem that is encountered in some LDAP deployments.

- Where Delta-syncrepl comes in:

Delta-syncrepl, a changelog-based variant of syncrepl, is designed to address situations like the one described above. Delta-syncrepl works by maintaining a changelog of a selectable depth in a separate database on the provider. The replication consumer checks the changelog for the changes it needs and, as long as the changelog contains the needed changes, the consumer fetches the changes from the changelog and applies them to its database. If, however, a replica is too far out of sync (or completely empty), conventional syncrepl is used to bring it up to date and replication then switches back to the delta-syncrepl mode.

Note: since the database state is stored in both the changelog DB and the main DB on the provider, it is important to backup/restore both the changelog DB and the main DB using slapcat/slapadd when restoring a DB or copying it to another machine.

For configuration, please see the Delta-syncrepl section.

18.2.2. N-Way Multi-Master replication

Multi-Master replication is a replication technique using Syncrepl to replicate data to multiple provider ("Master") Directory servers.

18.2.2.1. Valid Arguments for Multi-Master replication

- If any provider fails, other providers will continue to accept updates
- Avoids a single point of failure
- Providers can be located in several physical sites i.e. distributed across the network/globe.
- Good for Automatic failover/High Availability

18.2.2.2. Invalid Arguments for Multi-Master replication

(These are often claimed to be advantages of Multi-Master replication but those claims are false):

- It has **NOTHING** to do with load balancing
- Providers **must** propagate writes to **all** the other servers, which means the network traffic and write load spreads across all of the servers the same as for single-master.
- Server utilization and performance are at best identical for Multi-Master and Single-Master replication; at worst Single-Master is superior because indexing can be tuned differently to optimize for the different usage patterns between the provider and the consumers.

18.2.2.3. Arguments against Multi-Master replication

- Breaks the data consistency guarantees of the directory model
- http://www.openldap.org/faq/data/cache/1240.html
- If connectivity with a provider is lost because of a network partition, then "automatic failover" can just compound the problem
- Typically, a particular machine cannot distinguish between losing contact with a peer because that peer crashed, or because the network link has failed
- If a network is partitioned and multiple clients start writing to each of the "masters" then reconciliation will be a pain; it may be best to simply deny writes to the clients that are partitioned from the single provider

For configuration, please see the N-Way Multi-Master section below

18.2.3. MirrorMode replication

MirrorMode is a hybrid configuration that provides all of the consistency guarantees of single-master replication, while also providing the high availability of multi-master. In MirrorMode two providers are set up to replicate from each other (as a multi-master configuration), but an external frontend is employed to direct all writes to only one of the two servers. The second provider will only be used for writes if the first provider crashes, at which point the frontend will switch to directing all writes to the second provider. When a crashed provider is repaired and restarted it will automatically catch up to any changes on the running provider and resync.

18.2.3.1. Arguments for MirrorMode

- Provides a high-availability (HA) solution for directory writes (replicas handle reads)
- As long as one provider is operational, writes can safely be accepted
- Provider nodes replicate from each other, so they are always up to date and can be ready to take over (hot standby)
- Syncrepl also allows the provider nodes to re-synchronize after any downtime

18.2.3.2. Arguments against MirrorMode

- MirrorMode is not what is termed as a Multi-Master solution. This is because writes have to go to just one of the mirror nodes at a time
- MirrorMode can be termed as Active-Active Hot-Standby, therefore an external server (slapd in proxy mode) or device (hardware load balancer) is needed to manage which provider is currently active
- Backups are managed slightly differently
  - If backing up the Berkeley database itself and periodically backing up the transaction log files, then the same member of the mirror pair needs to be used to collect logfiles until the next database backup is taken

For configuration, please see the MirrorMode section below

18.2.4. Syncrepl Proxy Mode

While the LDAP Sync protocol supports both pull- and push-based replication, the push mode (refreshAndPersist) must still be initiated from the consumer before the provider can begin pushing changes. In some network configurations, particularly where firewalls restrict the direction in which connections can be made, a provider-initiated push mode may be needed.

This mode can be configured with the aid of the LDAP Backend (Backends and *slapd-ldap(8)*). Instead of running the syncrepl engine on the actual consumer, a slapd-ldap proxy is set up near (or collocated with) the provider that points to the consumer, and the syncrepl engine runs on the proxy.

For configuration, please see the Syncrepl Proxy section.

18.2.4.1. Replacing Slurpd

The old *slurpd* mechanism only operated in provider-initiated push mode. Slurpd replication was deprecated in favor of Syncrepl replication and has been completely removed from OpenLDAP 2.4.

The slurpd daemon was the original replication mechanism inherited from UMich's LDAP and operated in push mode: the master pushed changes to the slaves. It was replaced for many reasons, in brief:

- It was not reliable
  - It was extremely sensitive to the ordering of records in the replog
  - It could easily go out of sync, at which point manual intervention was required to resync the slave database with the master directory
  - It wasn't very tolerant of unavailable servers. If a slave went down for a long time, the replog could grow to a size that was too large for slurpd to process
- It only worked in push mode
- It required stopping and restarting the master to add new slaves
- It only supported single master replication

Syncrepl has none of those weaknesses:

- Syncrepl is self-synchronizing; you can start with a consumer database in any state from totally empty to fully synced and it will automatically do the right thing to achieve and maintain synchronization
  - It is completely insensitive to the order in which changes occur
  - It guarantees convergence between the consumer and the provider content without manual intervention
  - It can resynchronize regardless of how long a consumer stays out of contact with the provider
- Syncrepl can operate in either direction
- Consumers can be added at any time without touching anything on the provider
- Multi-master replication is supported

18.3. Configuring the different replication types

18.3.1. Syncrepl

18.3.1.1. Syncrepl configuration

Because syncrepl is a consumer-side replication engine, the syncrepl specification is defined in *slapd.conf*(5) of the consumer server, not in the provider server's configuration file. The initial loading of the replica content can be performed either by starting the syncrepl engine with no synchronization cookie or by populating the consumer replica by loading an LDIF file dumped as a backup at the provider.

When loading from a backup, it is not required to perform the initial loading from the up-to-date backup of the provider content. The syncrepl engine will automatically synchronize the initial consumer replica to the current provider content. As a result, it is not required to stop the provider server in order to avoid the replica inconsistency caused by the updates to the provider content during the content backup and loading process.

When replicating a large scale directory, especially in a bandwidth constrained environment, it is advised to load the consumer replica from a backup instead of performing a full initial load using syncrepl.

18.3.1.2. Set up the provider slapd

The provider is implemented as an overlay, so the overlay itself must first be configured in *slapd.conf*(5) before it can be used. The provider has only two configuration directives, for setting checkpoints on the `contextCSN` and for configuring the session log. Because the LDAP Sync search is subject to access control, proper access control privileges should be set up for the replicated content.

The `contextCSN` checkpoint is configured by the

```
syncprov-checkpoint <ops> <minutes>
```

directive. Checkpoints are only tested after successful write operations. If *<ops>* operations or more than *<minutes>* time has passed since the last checkpoint, a new checkpoint is performed.

The session log is configured by the

```
syncprov-sessionlog <size>
```

directive, where *<size>* is the maximum number of session log entries the session log can record. When a session log is configured, it is automatically used for all LDAP Sync searches within the database.

Note that using the session log requires searching on the *entryUUID* attribute. Setting an eq index on this attribute will greatly benefit the performance of the session log on the provider.

A more complete example of the *slapd.conf*(5) content is thus:

```
database mdb
maxsize 1073741824
suffix dc=Example,dc=com
rootdn dc=Example,dc=com
directory /var/ldap/db
index objectclass,entryCSN,entryUUID eq

overlay syncprov
syncprov-checkpoint 100 10
syncprov-sessionlog 100
```

18.3.1.3. Set up the consumer slapd

The syncrepl replication is specified in the database section of *slapd.conf*(5) for the replica context. The syncrepl engine is backend independent and the directive can be defined with any database type.

```
database mdb
maxsize 1073741824
suffix dc=Example,dc=com
rootdn dc=Example,dc=com
directory /var/ldap/db
index objectclass,entryCSN,entryUUID eq

syncrepl rid=123
        provider=ldap://provider.example.com:389
        type=refreshOnly
        interval=01:00:00:00
        searchbase="dc=example,dc=com"
        filter="(objectClass=organizationalPerson)"
        scope=sub
        attrs="cn,sn,ou,telephoneNumber,title,l"
        schemachecking=off
        bindmethod=simple
        binddn="cn=syncuser,dc=example,dc=com"
        credentials=secret
```

In this example, the consumer will connect to the provider *slapd*(8) at port 389 of ldap://provider.example.com to perform a polling (*refreshOnly*) mode of synchronization once a day. It will bind as `cn=syncuser,dc=example,dc=com` using simple authentication with password "secret". Note that the access control privilege of `cn=syncuser,dc=example,dc=com` should be set appropriately in the provider to retrieve the desired replication content. Also the search limits must be high enough on the provider to allow the syncuser to retrieve a complete copy of the requested content. The consumer uses the rootdn to write to its database so it always has full permissions to write all content.

The synchronization search in the above example will search for the entries whose objectClass is organizationalPerson in the

entire subtree rooted at `dc=example,dc=com`. The requested attributes are `cn`, `sn`, `ou`, `telephoneNumber`, `title`, and `l`. The schema checking is turned off, so that the consumer *slapd*(8) will not enforce entry schema checking when it processes updates from the provider *slapd*(8).

For more detailed information on the syncrepl directive, see the syncrepl section of The slapd Configuration File chapter of this admin guide.

18.3.1.4. Start the provider and the consumer slapd

The provider *slapd*(8) is not required to be restarted. *contextCSN* is automatically generated as needed: it might be originally contained in the LDIF file, generated by *slapadd* (8), generated upon changes in the context, or generated when the first LDAP Sync search arrives at the provider. If an LDIF file is being loaded which did not previously contain the *contextCSN*, the *-w* option should be used with *slapadd* (8) to cause it to be generated. This will allow the server to startup a little quicker the first time it runs.

When starting a consumer *slapd*(8), it is possible to provide a synchronization cookie as the *-c cookie* command line option in order to start the synchronization from a specific state. The cookie is a comma separated list of name=value pairs. Currently supported syncrepl cookie fields are *csn=<csn>* and *rid=<rid>*. *<csn>* represents the current synchronization state of the consumer replica. *<rid>* identifies a consumer replica locally within the consumer server. It is used to relate the cookie to the syncrepl definition in *slapd.conf*(5) which has the matching replica identifier. The *<rid>* must have no more than 3 decimal digits. The command line cookie overrides the synchronization cookie stored in the consumer replica database.

18.3.2. Delta-syncrepl

18.3.2.1. Delta-syncrepl Provider configuration

Setting up delta-syncrepl requires configuration changes on both the master and replica servers:

```
# Give the replica DN unlimited read access.  This ACL needs to be
# merged with other ACL statements, and/or moved within the scope
# of a database.  The "by * break" portion causes evaluation of
# subsequent rules.  See slapd.access(5) for details.
access to *
   by dn.base="cn=replicator,dc=symas,dc=com" read
   by * break

# Set the module path location
modulepath /opt/symas/lib/openldap

# Load the hdb backend
moduleload back_hdb.la

# Load the accesslog overlay
moduleload accesslog.la

#Load the syncprov overlay
moduleload syncprov.la

# Accesslog database definitions
database hdb
suffix cn=accesslog
directory /db/accesslog
rootdn cn=accesslog
index default eq
index entryCSN,objectClass,reqEnd,reqResult,reqStart

overlay syncprov
syncprov-nopresent TRUE
syncprov-reloadhint TRUE

# Let the replica DN have limitless searches
limits dn.exact="cn=replicator,dc=symas,dc=com" time.soft=unlimited time.hard=unlimited size.soft=unlimited size.hard=unlimited

# Primary database definitions
database hdb
suffix "dc=symas,dc=com"
rootdn "cn=manager,dc=symas,dc=com"

## Whatever other configuration options are desired

# syncprov specific indexing
index entryCSN eq
index entryUUID eq

# syncrepl Provider for primary db
overlay syncprov
syncprov-checkpoint 1000 60

# accesslog overlay definitions for primary db
overlay accesslog
logdb cn=accesslog
logops writes
logsuccess TRUE
# scan the accesslog DB every day, and purge entries older than 7 days
logpurge 07+00:00 01+00:00

# Let the replica DN have limitless searches
limits dn.exact="cn=replicator,dc=symas,dc=com" time.soft=unlimited time.hard=unlimited size.soft=unlimited size.hard=unlimited
```

For more information, always consult the relevant man pages (*slapo-accesslog*(5) and *slapd.conf*(5))

18.3.2.2. Delta-syncrepl Consumer configuration

```
# Replica database configuration
database hdb
```

```
suffix "dc=symas,dc=com"
rootdn "cn=manager,dc=symas,dc=com"

## Whatever other configuration bits for the replica, like indexing
## that you want

# syncrepl specific indices
index entryUUID eq

# syncrepl directives
syncrepl  rid=0
          provider=ldap://ldapmaster.symas.com:389
          bindmethod=simple
          binddn="cn=replicator,dc=symas,dc=com"
          credentials=secret
          searchbase="dc=symas,dc=com"
          logbase="cn=accesslog"
          logfilter="(&(objectClass=auditWriteObject)(reqResult=0))"
          schemachecking=on
          type=refreshAndPersist
          retry="60 +"
          syncdata=accesslog

# Refer updates to the master
updateref               ldap://ldapmaster.symas.com
```

The above configuration assumes that you have a replicator identity defined in your database that can be used to bind to the provider. In addition, all of the databases (primary, replica, and the accesslog storage database) should also have properly tuned *DB_CONFIG* files that meet your needs.

18.3.3. N-Way Multi-Master

For the following example we will be using 3 Master nodes. Keeping in line with **test050-syncrepl-multimaster** of the OpenLDAP test suite, we will be configuring *slapd(8)* via **cn=config**

This sets up the config database:

```
dn: cn=config
objectClass: olcGlobal
cn: config
olcServerID: 1

dn: olcDatabase={0}config,cn=config
objectClass: olcDatabaseConfig
olcDatabase: {0}config
olcRootPW: secret
```

second and third servers will have a different olcServerID obviously:

```
dn: cn=config
objectClass: olcGlobal
cn: config
olcServerID: 2

dn: olcDatabase={0}config,cn=config
objectClass: olcDatabaseConfig
olcDatabase: {0}config
olcRootPW: secret
```

This sets up syncrepl as a provider (since these are all masters):

```
dn: cn=module,cn=config
objectClass: olcModuleList
cn: module
olcModulePath: /usr/local/libexec/openldap
olcModuleLoad: syncprov.la
```

Now we setup the first Master Node (replace $URI1, $URI2 and $URI3 etc. with your actual ldap urls):

```
dn: cn=config
changetype: modify
replace: olcServerID
olcServerID: 1 $URI1
olcServerID: 2 $URI2
olcServerID: 3 $URI3

dn: olcOverlay=syncprov,olcDatabase={0}config,cn=config
changetype: add
objectClass: olcOverlayConfig
objectClass: olcSyncProvConfig
olcOverlay: syncprov

dn: olcDatabase={0}config,cn=config
changetype: modify
add: olcSyncRepl
olcSyncRepl: rid=001 provider=$URI1 binddn="cn=config" bindmethod=simple
  credentials=secret searchbase="cn=config" type=refreshAndPersist
  retry="5 5 300 5" timeout=1
olcSyncRepl: rid=002 provider=$URI2 binddn="cn=config" bindmethod=simple
  credentials=secret searchbase="cn=config" type=refreshAndPersist
  retry="5 5 300 5" timeout=1
olcSyncRepl: rid=003 provider=$URI3 binddn="cn=config" bindmethod=simple
  credentials=secret searchbase="cn=config" type=refreshAndPersist
  retry="5 5 300 5" timeout=1
-
add: olcMirrorMode
olcMirrorMode: TRUE
```

Now start up the Master and a consumer/s, also add the above LDIF to the first consumer, second consumer etc. It will then replicate **cn=config**. You now have N-Way Multimaster on the config database.

We still have to replicate the actual data, not just the config, so add to the master (all active and configured consumers/masters will pull down this config, as they are all syncing). Also, replace all *${}* variables with whatever is applicable to your setup:

```
dn: olcDatabase={1}$BACKEND,cn=config
objectClass: olcDatabaseConfig
objectClass: olc${BACKEND}Config
olcDatabase: {1}$BACKEND
olcSuffix: $BASEDN
olcDbDirectory: ./db
olcRootDN: $MANAGERDN
olcRootPW: $PASSWD
olcLimits: dn.exact="$MANAGERDN" time.soft=unlimited time.hard=unlimited size.soft=unlimited size.hard=unlimited
olcSyncRepl: rid=004 provider=$URI1 binddn="$MANAGERDN" bindmethod=simple
  credentials=$PASSWD searchbase="$BASEDN" type=refreshOnly
  interval=00:00:00:10 retry="5 5 300 5" timeout=1
olcSyncRepl: rid=005 provider=$URI2 binddn="$MANAGERDN" bindmethod=simple
  credentials=$PASSWD searchbase="$BASEDN" type=refreshOnly
  interval=00:00:00:10 retry="5 5 300 5" timeout=1
olcSyncRepl: rid=006 provider=$URI3 binddn="$MANAGERDN" bindmethod=simple
  credentials=$PASSWD searchbase="$BASEDN" type=refreshOnly
  interval=00:00:00:10 retry="5 5 300 5" timeout=1
olcMirrorMode: TRUE

dn: olcOverlay=syncprov,olcDatabase={1}${BACKEND},cn=config
changetype: add
objectClass: olcOverlayConfig
objectClass: olcSyncProvConfig
olcOverlay: syncprov
```

Note: All of your servers' clocks must be tightly synchronized using e.g. NTP http://www.ntp.org/, atomic clock, or some other reliable time reference.

Note: As stated in *slapd-config*(5), URLs specified in *olcSyncRepl* directives are the URLs of the servers from which to replicate. These must exactly match the URLs *slapd* listens on (*-h* in Command-Line Options). Otherwise slapd may attempt to replicate from itself, causing a loop.

18.3.4. MirrorMode

MirrorMode configuration is actually very easy. If you have ever setup a normal slapd syncrepl provider, then the only change is the following two directives:

```
mirrormode  on
serverID    1
```

Note: You need to make sure that the *serverID* of each mirror node is different and add it as a global configuration option.

18.3.4.1. Mirror Node Configuration

The first step is to configure the syncrepl provider the same as in the Set up the provider slapd section.

Here's a specific cut down example using LDAP Sync Replication in *refreshAndPersist* mode:

MirrorMode node 1:

```
# Global section
serverID    1
# database section

# syncrepl directive
syncrepl      rid=001
              provider=ldap://ldap-sid2.example.com
              bindmethod=simple
              binddn="cn=mirrormode,dc=example,dc=com"
              credentials=mirrormode
              searchbase="dc=example,dc=com"
              schemachecking=on
              type=refreshAndPersist
              retry="60 +"

mirrormode on
```

MirrorMode node 2:

```
# Global section
serverID    2
# database section

# syncrepl directive
syncrepl      rid=001
              provider=ldap://ldap-sid1.example.com
              bindmethod=simple
              binddn="cn=mirrormode,dc=example,dc=com"
              credentials=mirrormode
              searchbase="dc=example,dc=com"
              schemachecking=on
              type=refreshAndPersist
              retry="60 +"

mirrormode on
```

It's simple really; each MirrorMode node is setup **exactly** the same, except that the *serverID* is unique, and each consumer is pointed to the other server.

18.3.4.1.1. Failover Configuration

There are generally 2 choices for this; 1. Hardware proxies/load-balancing or dedicated proxy software, 2. using a Back-LDAP proxy as a syncrepl provider

A typical enterprise example might be:

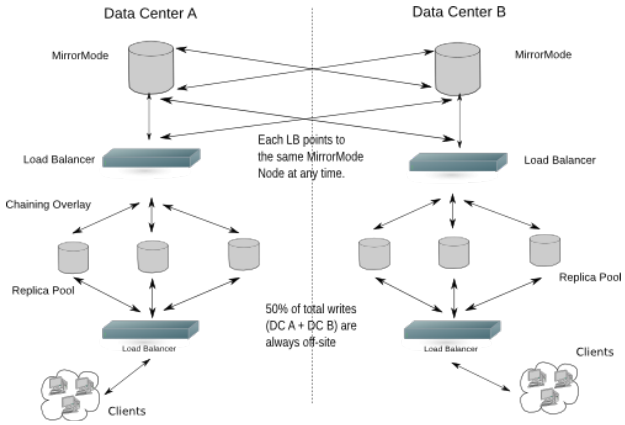

Figure X.Y: MirrorMode in a Dual Data Center Configuration

18.3.4.1.2. Normal Consumer Configuration

This is exactly the same as the Set up the consumer slapd section. It can either setup in normal syncrepl replication mode, or in delta-syncrepl replication mode.

18.3.4.2. MirrorMode Summary

You will now have a directory architecture that provides all of the consistency guarantees of single-master replication, while also providing the high availability of multi-master replication.

18.3.5. Syncrepl Proxy

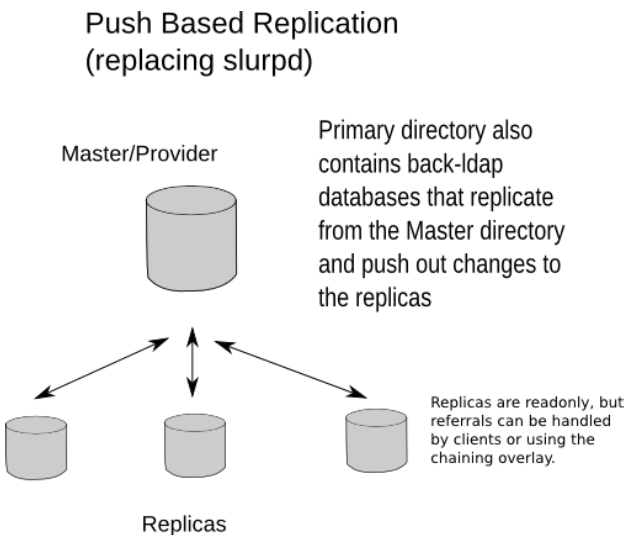

Figure X.Y: Replacing slurpd

The following example is for a self-contained push-based replication solution:

```
########################################################################
# Standard OpenLDAP Master/Provider
########################################################################

include     /usr/local/etc/openldap/schema/core.schema
include     /usr/local/etc/openldap/schema/cosine.schema
include     /usr/local/etc/openldap/schema/nis.schema
include     /usr/local/etc/openldap/schema/inetorgperson.schema

include     /usr/local/etc/openldap/slapd.acl

modulepath  /usr/local/libexec/openldap
moduleload  back_hdb.la
moduleload  syncprov.la
moduleload  back_monitor.la
moduleload  back_ldap.la

pidfile     /usr/local/var/slapd.pid
```

```
argsfile    /usr/local/var/slapd.args

loglevel    sync stats

database    hdb
suffix      "dc=suretecsystems,dc=com"
directory   /usr/local/var/openldap-data

checkpoint      1024 5
cachesize       10000
idlcachesize    10000

index       objectClass eq
# rest of indexes
index       default     sub

rootdn          "cn=admin,dc=suretecsystems,dc=com"
rootpw          testing

# syncprov specific indexing
index entryCSN eq
index entryUUID eq

# syncrepl Provider for primary db
overlay syncprov
syncprov-checkpoint 1000 60

# Let the replica DN have limitless searches
limits dn.exact="cn=replicator,dc=suretecsystems,dc=com" time.soft=unlimited time.hard=unlimited size.soft=unlimited size.hard=unlimited

database    monitor

database    config
rootpw          testing

#######################################################################
# Consumer Proxy that pulls in data via Syncrepl and pushes out via slapd-ldap
#######################################################################

database        ldap
# ignore conflicts with other databases, as we need to push out to same suffix
hidden              on
suffix          "dc=suretecsystems,dc=com"
rootdn          "cn=slapd-ldap"
uri             ldap://localhost:9012/

lastmod         on

# We don't need any access to this DSA
restrict        all

acl-bind        bindmethod=simple
                binddn="cn=replicator,dc=suretecsystems,dc=com"
                credentials=testing

syncrepl        rid=001
                provider=ldap://localhost:9011/
                binddn="cn=replicator,dc=suretecsystems,dc=com"
                bindmethod=simple
                credentials=testing
                searchbase="dc=suretecsystems,dc=com"
                type=refreshAndPersist
                retry="5 5 300 5"

overlay         syncprov
```

A replica configuration for this type of setup could be:

```
#######################################################################
# Standard OpenLDAP Slave without Syncrepl
#######################################################################

include     /usr/local/etc/openldap/schema/core.schema
include     /usr/local/etc/openldap/schema/cosine.schema
include     /usr/local/etc/openldap/schema/nis.schema
include     /usr/local/etc/openldap/schema/inetorgperson.schema

include     /usr/local/etc/openldap/slapd.acl

modulepath  /usr/local/libexec/openldap
moduleload  back_hdb.la
moduleload  syncprov.la
moduleload  back_monitor.la
moduleload  back_ldap.la

pidfile     /usr/local/var/slapd.pid
argsfile    /usr/local/var/slapd.args

loglevel    sync stats

database    hdb
suffix      "dc=suretecsystems,dc=com"
directory   /usr/local/var/openldap-slave/data

checkpoint      1024 5
cachesize       10000
idlcachesize    10000

index       objectClass eq
# rest of indexes
index       default     sub

rootdn          "cn=admin,dc=suretecsystems,dc=com"
rootpw          testing
```

```
# Let the replica DN have limitless searches
limits dn.exact="cn=replicator,dc=suretecsystems,dc=com" time.soft=unlimited time.hard=unlimited size.soft=unlimited size.hard=unlimited

updatedn "cn=replicator,dc=suretecsystems,dc=com"

# Refer updates to the master
updateref   ldap://localhost:9011

database    monitor

database    config
rootpw          testing
```

You can see we use the *updatedn* directive here and example ACLs (`usr/local/etc/openldap/slapd.acl`) for this could be:

```
# Give the replica DN unlimited read access.  This ACL may need to be
# merged with other ACL statements.

access to *
     by dn.base="cn=replicator,dc=suretecsystems,dc=com" write
     by * break

access to dn.base=""
        by * read

access to dn.base="cn=Subschema"
        by * read

access to dn.subtree="cn=Monitor"
    by dn.exact="uid=admin,dc=suretecsystems,dc=com" write
    by users read
    by * none

access to *
        by self write
        by * read
```

In order to support more replicas, just add more *database ldap* sections and increment the *syncrepl rid* number accordingly.

Note: You must populate the Master and Slave directories with the same data, unlike when using normal Syncrepl

---

If you do not have access to modify the master directory configuration you can configure a standalone ldap proxy, which might look like:

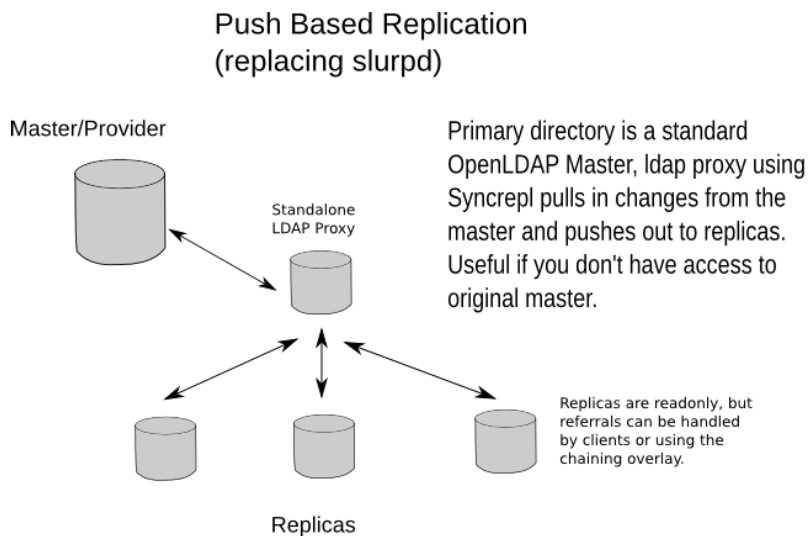

Figure X.Y: Replacing slurpd with a standalone version

The following configuration is an example of a standalone LDAP Proxy:

```
include     /usr/local/etc/openldap/schema/core.schema
include     /usr/local/etc/openldap/schema/cosine.schema
include     /usr/local/etc/openldap/schema/nis.schema
include     /usr/local/etc/openldap/schema/inetorgperson.schema

include     /usr/local/etc/openldap/slapd.acl

modulepath  /usr/local/libexec/openldap
moduleload  syncprov.la
moduleload  back_ldap.la

#######################################################################
# Consumer Proxy that pulls in data via Syncrepl and pushes out via slapd-ldap
#######################################################################

database        ldap
# ignore conflicts with other databases, as we need to push out to same suffix
hidden              on
suffix          "dc=suretecsystems,dc=com"
rootdn          "cn=slapd-ldap"
uri             ldap://localhost:9012/
```

```
        lastmod         on

        # We don't need any access to this DSA
        restrict        all

        acl-bind        bindmethod=simple
                        binddn="cn=replicator,dc=suretecsystems,dc=com"
                        credentials=testing

        syncrepl        rid=001
                        provider=ldap://localhost:9011/
                        binddn="cn=replicator,dc=suretecsystems,dc=com"
                        bindmethod=simple
                        credentials=testing
                        searchbase="dc=suretecsystems,dc=com"
                        type=refreshAndPersist
                        retry="5 5 300 5"

        overlay         syncprov
```

As you can see, you can let your imagination go wild using Syncrepl and *slapd-ldap(8)* tailoring your replication to fit your specific network topology.

Contents | Parent Topic | Previous Topic | Next Topic
Home | Catalog