Configuration

This guide shows you how to configure DS server features.

HTTP

Access DS over HTTP.

LDAP

Access DS over LDAP.

Storage

Manage DS data.

Indexes

Index DS data.

Replication

Replicate DS data.

LDAP Proxy

Configure proxy features.

ForgeRock® Identity Platform serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

The ForgeRock® Common REST API works across the platform to provide common ways to access web resources and collections of resources.

HTTP Access

Set the HTTP Port

The following steps demonstrate how to set up an HTTP port if none was configured at setup time with the `--httpPort` option:

1. Create an HTTP connection handler:

```
$ dsconfig \
 create-connection-handler \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name HTTP \
 --type http \
 --set enabled:true \
 --set listen-port:8080 \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

2. Enable an HTTP access log.

   a. The following command enables JSON-based HTTP access logging:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --publisher-name "Json File-Based HTTP Access Logger"
\
 --set enabled:true \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore
\
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

b. The following command enables HTTP access logging:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --publisher-name "File-Based HTTP Access Logger" \
 --set enabled:true \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore
\
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

3. After you set up an HTTP port, enable an HTTP endpoint.

   For details, see Configure HTTP User APIs or Use Administrative APIs.

Set the HTTPS Port

At setup time use the `--httpsPort` option.

Later, follow these steps to set up an HTTPS port:

1. Create an HTTPS connection handler.

   The following example sets the port to `8443` and uses the default server certificate:

```
$ dsconfig \
 create-connection-handler \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name HTTPS \
 --type http \
 --set enabled:true \
 --set listen-port:8443 \
 --set use-ssl:true \
 --set key-manager-provider:PKCS12 \
 --set trust-manager-provider:"JVM Trust Manager" \
```

```
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

If the key manager provider has multiple key pairs that DS could use for TLS, where the secret key was generated with the same key algorithm, such as `EC` or `RSA`, you can specify which key pairs to use with the `--set ssl-cert-nickname:`*`server-cert`* option. The *server-cert* is the certificate alias of the key pair. This option is not necessary if there is only one server key pair, or if each secret key was generated with a different key algorithm.

2. Enable the HTTP access log.

   a. The following command enables JSON-based HTTP access logging:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --publisher-name "Json File-Based HTTP Access Logger"
\
 --set enabled:true \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore
\
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

   b. The following command enables HTTP access logging:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --publisher-name "File-Based HTTP Access Logger" \
 --set enabled:true \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore
\
```

```
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

3. If the deployment requires SSL client authentication, set the properties `ssl-client-auth-policy` and `trust-manager-provider` appropriately.
4. After you set up an HTTPS port, enable an HTTP endpoint.

   For details, see Configure HTTP User APIs, or Use Administrative APIs.

Configure HTTP User APIs

The way directory data appears to client applications is configurable. You configure a Rest2ldap endpoint for each HTTP API to user data.

A Rest2ldap mapping file defines how JSON resources map to LDAP entries. The default mapping file is `/path/to/opendj/config/rest2ldap/endpoints/api/example-v1.json`. The default mapping works with Example.com data from the evaluation setup profile.

Edit or add mapping files for your own APIs. For details, see REST to LDAP Reference.

If you have set up a directory server with the `ds-evaluation` profile, you can skip the first two steps:

1. If necessary, change the properties of the default Rest2ldap endpoint, or create a new endpoint.

   The default Rest2ldap HTTP endpoint is named `/api` after its `base-path`. The `base-path` must be the same as the name, and is read-only after creation. By default, the `/api` endpoint requires authentication.

   The following example enables the default `/api` endpoint:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name /api \
 --set authorization-mechanism:"HTTP Basic" \
 --set config-directory:config/rest2ldap/endpoints/api \
 --set enabled:true \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

```
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

Alternatively, you can create another Rest2ldap endpoint to expose a different HTTP API, or to publish data under an alternative base path, such as `/rest`:

```
$ dsconfig \
 create-http-endpoint \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name /rest \
 --type rest2ldap-endpoint \
 --set authorization-mechanism:"HTTP Basic" \
 --set config-directory:config/rest2ldap/endpoints/api \
 --set enabled:true \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

2. If necessary, adjust the endpoint configuration to use an alternative HTTP authorization mechanism.

   By default, the Rest2ldap endpoint maps HTTP Basic authentication to LDAP authentication. Change the `authorization-mechanism` setting as necessary. For details, see Configure HTTP Authorization.

3. Try reading a resource.

   The following example generates the CA certificate in PEM format from the server deployment key and password. It uses the CA certificate to trust the server certificate. A CA certificate is only necessary if the CA is not well-known:

```
$ dskeymgr \
 export-ca-cert \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --outputFile ca-cert.pem

$ curl \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 https://localhost:8443/api/users/bjensen?
_prettyPrint=true
```

```
{
  "_id" : "bjensen",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : { },
  "userName" : "bjensen@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1862",
    "emailAddress" : "bjensen@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen",
  "manager" : {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  }
}
```

Configure HTTP Authorization

HTTP authorization mechanisms map HTTP credentials to LDAP credentials.

Multiple HTTP authorization mechanisms can be enabled simultaneously. You can assign different mechanisms to Rest2ldap endpoints and to the Admin endpoint.

By default, these HTTP authorization mechanisms are supported:

HTTP Anonymous
: Process anonymous HTTP requests, optionally binding with a specified DN.

  If no bind DN is specified (default), anonymous LDAP requests are used.

  This mechanism is enabled by default.

HTTP Basic (enabled by default)
: Process HTTP Basic authentication requests by mapping the HTTP Basic identity to a user's directory account.

By default, the exact match identity mapper with its default configuration is used to map the HTTP Basic user name to an LDAP `uid` . The DS server then searches in all local public naming contexts to find the user's entry based in the `uid` value. For details, see Identity Mappers.

HTTP OAuth2 CTS

Process OAuth 2.0 requests as a resource server, acting as an AM Core Token Service (CTS) store.

When the client bearing an OAuth 2.0 access token presents the token to access the JSON resource, the server tries to resolve the access token against the CTS data that it serves for AM. If the access token resolves correctly (is found in the CTS data and has not expired), the DS server extracts the user identity and OAuth 2.0 scopes. If the required scopes are present and the token is valid, it maps the user identity to a directory account.

This mechanism makes it possible to resolve access tokens by making an internal request, avoiding a request to AM. *This mechanism does not ensure that the token requested will have already been replicated to the replica where the request is routed.*

AM's CTS store is constrained to a specific layout. The `authzid-json-pointer` must use `userName/0` for the user identifier.

HTTP OAuth2 OpenAM

Process OAuth 2.0 requests as a resource server, sending requests to AM for access token resolution.

When the client bearing an OAuth 2.0 access token presents the token to access the JSON resource, the directory service requests token information from AM. If the access token is valid, the DS server extracts the user identity and OAuth 2.0 scopes. If the required scopes are present, it maps the user identity to a directory account.

Access token resolution requests should be sent over HTTPS. You can configure a truststore manager if necessary to trust the authorization server certificate, and a keystore manager to obtain the DS server certificate if the authorization server requires mutual authentication.

HTTP OAuth2 Token Introspection (RFC7662)

Handle OAuth 2.0 requests as a resource server, sending requests to an RFC 7662-compliant authorization server for access token resolution.

The DS server must be registered as a client of the authorization server.

When the client bearing an OAuth 2.0 access token presents the token to access the JSON resource, the DS server requests token introspection from the authorization server. If the access token is valid, the DS server extracts the user identity and OAuth 2.0 scopes. If the required scopes are present, it maps the user identity to a directory account.

Access token resolution requests should be sent over HTTPS. You can configure a truststore manager if necessary to trust the authorization server certificate, and a keystore manager to obtain the DS server certificate if the authorization server requires mutual authentication.

NOTE

The HTTP OAuth2 File mechanism is an internal interface intended for testing, and not supported for production use.

When more than one authentication mechanism is specified, mechanisms are applied in the following order:

- If the client request has an `Authorization` header, and an OAuth 2.0 mechanism is specified, the server attempts to apply the OAuth 2.0 mechanism.
- If the client request has an `Authorization` header, or has the custom credentials headers specified in the configuration, and an HTTP Basic mechanism is specified, the server attempts to apply the Basic Auth mechanism.
- Otherwise, if an HTTP anonymous mechanism is specified, and none of the previous mechanisms apply, the server attempts to apply the mechanism for anonymous HTTP requests.

There are many possibilities when configuring HTTP authorization mechanisms. *This procedure shows only one OAuth 2.0 example.*

The example below uses settings as listed in the following table. When using secure connections, make sure the servers can trust each other's certificates. Download ForgeRock Access Management software from the ForgeRock BackStage download site :

Setting	Value
OpenAM URL	`https://am.example.com:8443/openam` (When using HTTPS, make sure DS can trust the AM certificate.)
Authorization server endpoint	`/oauth2/tokeninfo` (top-level realm)
Identity repository	DS server configured by the examples that follow.
OAuth 2.0 client ID	`myClientID`
OAuth 2.0 client secret	`password`
OAuth 2.0 client scopes	`read`, `uid`, `write`

Setting	Value
Rest2ldap configuration	Default settings. See Configure HTTP User APIs.

Read the ForgeRock Access Management documentation if necessary to install and configure AM. Then follow these steps to try the demonstration:

1. Update the default HTTP OAuth2 OpenAM configuration:

```
$ dsconfig \
 set-http-authorization-mechanism-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --mechanism-name "HTTP OAuth2 OpenAM" \
 --set enabled:true \
 --set token-info-
url:https://am.example.com:8443/openam/oauth2/tokeninfo \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

2. Update the default Rest2ldap endpoint configuration to use HTTP OAuth2 OpenAM as the authorization mechanism:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set authorization-mechanism:"HTTP OAuth2 OpenAM" \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

3. Obtain an access token with the appropriate scopes:

```
$ curl \
 --request POST \
```

```
 --user "myClientID:password" \
 --data
"grant_type=password&username=bjensen&password=hifalutin&s
cope=read%20uid%20write" \
 https://am.example.com:8443/openam/oauth2/access_token
{
 "access_token": "token-string",
 "scope": "uid read write",
 "token_type": "Bearer",
 "expires_in": 3599
}
```

Use HTTPS when obtaining access tokens.

4. Request a resource at the Rest2ldap endpoint using HTTP Bearer authentication with the access token:

```
$ curl \
 --header "Authorization: Bearer token-string" \
 --cacert ca-cert.pem \
 https://localhost:8443/api/users/bjensen?
_prettyPrint=true
{
  "_id": "bjensen",
  "_rev": "<revision>",
  "_schema": "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta": {},
  "userName": "bjensen@example.com",
  "displayName": ["Barbara Jensen", "Babs Jensen"],
  "name": {
    "givenName": "Barbara",
    "familyName": "Jensen"
  },
  "description": "Original description",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1862",
    "emailAddress": "bjensen@example.com"
  },
  "uidNumber": 1076,
  "gidNumber": 1000,
  "homeDirectory": "/home/bjensen",
  "manager": {
    "_id": "trigden",
    "displayName": "Torrey Rigden"
```

```
    }
}
```

Use HTTPS when presenting access tokens.

Use Administrative APIs

The APIs for configuring and monitoring DS servers are under the following endpoints:

/admin/config
: Provides a REST API to the server configuration under `cn=config`.

  By default, this endpoint is protected by the HTTP Basic authorization mechanism. Users reading and editing the configuration must have appropriate privileges, such as `config-read` and `config-write`.

  Each LDAP entry maps to a resource under `/admin/config`, with default values shown in the resource even if they are not set in the LDAP representation.

/alive
: Provides an endpoint to check whether the server is currently *alive*, meaning that its internal checks have not found any errors that would require administrative action.

  By default, this endpoint returns a status code to anonymous requests, and supports authenticated requests. For details, see Server is Alive (HTTP).

/healthy
: Provides an endpoint to check whether the server is currently *healthy*, meaning that it is alive and any replication delays are below a configurable threshold.

  By default, this endpoint returns a status code to anonymous requests, and supports authenticated requests. For details, see Server Health (HTTP).

/metrics/api
: Provides a read-only REST API to the server monitoring information under `cn=monitor`.

  By default, this endpoint is protected by the HTTP Basic authorization mechanism. Users reading monitoring information must have the `monitor-read` privilege.

  Each LDAP entry maps to a resource under `/metrics/api`.

/metrics/prometheus
: Provides an API to the server monitoring information for use with Prometheus monitoring software.

  By default, this endpoint is protected by the HTTP Basic authorization mechanism. Users reading monitoring information must have the `monitor-read` privilege.

To use the Admin endpoint APIs, follow these steps:

1. Grant users access to the endpoints as appropriate:

   a. For access to `/admin/config`, assign `config-read` or `config-write` privileges.

      The following example assigns the `config-read` privilege to Kirsten Vaughan:

      ```
      $ ldapmodify \
       --hostname localhost \
       --port 1636 \
       --useSsl \
       --usePkcs12TrustStore /path/to/opendj/config/keystore
      \
       --trustStorePassword:file
      /path/to/opendj/config/keystore.pin \
       --bindDN uid=admin \
       --bindPassword password << EOF
      dn: uid=kvaughan,ou=People,dc=example,dc=com
      changetype: modify
      add: ds-privilege-name
      ds-privilege-name: config-read
      EOF
      ```

   b. For access to `/metrics` endpoints, if no account for monitoring was created at setup time, assign the `monitor-read` privilege.

      The following example adds the `monitor-read` privilege to Kirsten Vaughan's entry:

      ```
      $ ldapmodify \
       --hostname localhost \
       --port 1636 \
       --useSsl \
       --usePkcs12TrustStore /path/to/opendj/config/keystore
      \
       --trustStorePassword:file
      /path/to/opendj/config/keystore.pin \
       --bindDN uid=admin \
       --bindPassword password << EOF
      dn: uid=kvaughan,ou=People,dc=example,dc=com
      changetype: modify
      add: ds-privilege-name
      ```

```
ds-privilege-name: monitor-read
EOF
```

For details, see Administrative Privileges.

2. Adjust the `authorization-mechanism` settings for the Admin endpoint.

   By default, the Admin endpoint uses the HTTP Basic authorization mechanism. The HTTP Basic authorization mechanism default configuration resolves the user identity extracted from the HTTP request to an LDAP user identity as follows:

   a. If the request has an `Authorization: Basic` header for HTTP Basic authentication, the server extracts the username and password.

   b. If the request has `X-OpenIDM-Username` and `X-OpenIDM-Password` headers, the server extracts the username and password.

   c. The server uses the default exact match identity mapper to search for a unique match between the username and the UID attribute value of an entry in the local public naming contexts of the DS server.

      In other words, in LDAP terms, it searches under all user data base DNs for `(uid=http-username)`. The username `kvaughan` maps to the example entry with DN `uid=kvaughan,ou=People,dc=example,dc=com`.

      For details, see Identity Mappers, and Configure HTTP Authorization.

3. Test access to the endpoint as an authorized user.

   The following example reads the Admin endpoint resource under `/admin/config`. The following example generates the CA certificate in PEM format from the server deployment key and password. It uses the CA certificate to trust the server certificate. A CA certificate is only necessary if the CA is not well-known:

```
$ dskeymgr \
 export-ca-cert \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --outputFile ca-cert.pem

$ curl \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 https://localhost:8443/admin/config/http-
endpoints/%2Fadmin?_prettyPrint=true

{
```

```
  "_id" : "/admin",
  "_rev" : "<revision>",
  "_schema" : "admin-endpoint",
  "java-class" :
"org.opends.server.protocols.http.rest2ldap.AdminEndpoint"
,
  "base-path" : "/admin",
  "enabled" : true,
  "authorization-mechanism" : "HTTP Basic"
}
```

Notice how the path to the resource in the example above, `/admin/config/http-endpoints/%2Fadmin`, corresponds to the DN of the entry under `cn=config`, which is `ds-cfg-base-path=/admin,cn=HTTP Endpoints,cn=config`.

The following example demonstrates reading everything under `/metrics/api`:

```
$ curl \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 https://localhost:8443/metrics/api?_queryFilter=true
```

LDAP Access

Set the LDAP Port

The reserved port number for LDAP is `389`. Most examples in the documentation use `1389`, which is accessible to non-privileged users:

1. The following example changes the LDAP port number to `11389`:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name LDAP \
 --set listen-port:11389 \
```

```
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

2. Restart the connection handler, and the change takes effect:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name LDAP \
 --set enabled:false \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 set-connection-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name LDAP \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

Enable StartTLS

StartTLS negotiations start on the unsecure LDAP port, and then protect communication with the client:

1. Activate StartTLS on the current LDAP port:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname localhost \
```

```
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name LDAP \
 --set allow-start-tls:true \
 --set key-manager-provider:PKCS12 \
 --set trust-manager-provider:"JVM Trust Manager" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

If the key manager provider has multiple key pairs that DS could use for TLS, where the secret key was generated with the same key algorithm, such as `EC` or `RSA`, you can specify which key pairs to use with the `--set ssl-cert-nickname:`*`server-cert`* option. The *server-cert* is the certificate alias of the key pair. This option is not necessary if there is only one server key pair, or if each secret key was generated with a different key algorithm.

The change takes effect. No need to restart the server.

Set the LDAPS port

At setup time, use the `--ldapsPort` option.

Later, follow these steps to set up an LDAPS port:

1. Configure the server to activate LDAPS access:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name LDAPS \
 --set enabled:true \
 --set listen-port:1636 \
 --set use-ssl:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

2. If the deployment requires SSL client authentication, set the `ssl-client-auth-policy` and `trust-manager-provider` properties appropriately.

Set the LDAPS Port

The reserved port number for LDAPS is `636`. Most examples in the documentation use `1636`, which is accessible to non-privileged users.

1. Change the port number using the **`dsconfig`** command.

   The following example changes the LDAPS port number to `11636`:

   ```
   $ dsconfig \
    set-connection-handler-prop \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --handler-name LDAPS \
    --set listen-port:11636 \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePassword:file
   /path/to/opendj/config/keystore.pin \
    --no-prompt
   ```

2. Restart the connection handler so the change takes effect.

   To restart the connection handler, you disable it, then enable it again:

   ```
   $ dsconfig \
    set-connection-handler-prop \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --handler-name LDAPS \
    --set enabled:false \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePassword:file
   /path/to/opendj/config/keystore.pin \
    --no-prompt

   $ dsconfig \
    set-connection-handler-prop \
   ```

```
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name LDAPS \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

LDIF File Access

The LDIF connection handler lets you change directory data by placing LDIF files in a file system directory. The DS server regularly polls for changes to the directory. The server deletes the LDIF file after making the changes.

1. Add the directory where you put LDIF to be processed:

   ```
   $ mkdir /path/to/opendj/config/auto-process-ldif
   ```

   This example uses the default value of the `ldif-directory` property.

2. Activate LDIF file access:

   ```
   $ dsconfig \
    set-connection-handler-prop \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --handler-name LDIF \
    --set enabled:true \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePassword:file
   /path/to/opendj/config/keystore.pin \
    --no-prompt
   ```

   The change takes effect immediately.

LDAP Schema

About LDAP Schema

Directory schema, described in RFC 4512, define the kinds of information you find in the directory, and how the information is related.

By default, DS servers conform strictly to LDAPv3 standards for schema definitions and syntax checking. This ensures that data stored is valid and properly formed. Unless your data uses only standard schema present in the server when you install, you must add additional schema definitions to account for the data specific to your applications.

DS servers include many standard schema definitions. You can update and extend schema definitions while DS servers are online. As a result, you can add new applications requiring additional data without stopping your directory service.

The examples that follow focus primarily on the following types of directory schema definitions:

- *Attribute type* definitions describe attributes of directory entries, such as `givenName` or `mail`.

  Here is an example of an attribute type definition:

  ```
  # Attribute type definition
  attributeTypes: ( 0.9.2342.19200300.100.1.3 NAME ( 'mail'
  'rfc822Mailbox' )
    EQUALITY caseIgnoreIA5Match SUBSTR
  caseIgnoreIA5SubstringsMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{256} X-ORIGIN 'RFC
  4524' )
  ```

  Attribute type definitions start with an OID, and a short name or names that are easier to remember. The attribute type definition can specify how attribute values should be collated for sorting, and what syntax they use.

  The X-ORIGIN is an extension to identify where the definition originated. When you define your own schema, provide an X-ORIGIN to help track versions of definitions.

  Attribute type definitions indicate whether the attribute is:

  - A *user attribute* intended to be modified by external applications.

    This is the default, and you can make it explicit with `USAGE userApplications`.

The user attributes that are required and permitted on each entry are defined by the entry's object classes. The server checks what the entry's object classes require and permit when updating user attributes.

- An *operational attribute* intended to be managed by the server for internal purposes.

  You can specify this with `USAGE directoryOperation`.

  The server does not check whether an operational attribute is allowed by an object class.

Attribute type definitions differentiate operational attributes with the following `USAGE` types:

- `USAGE directoryOperation` indicates a generic operational attribute.

  Use this type, for example, when creating a last login time attribute.
- `USAGE dSAOperation` indicates a DSA-specific operational attribute, meaning an operational attribute specific to the current server.
- `USAGE distributedOperation` indicates a DSA-shared operational attribute, meaning an operational attribute shared by multiple servers.

- *Object class* definitions identify the attribute types that an entry must have, and may have.

  Here is an example of an object class definition:

```
# Object class definition
objectClasses: ( 2.5.6.6 NAME 'person' SUP top STRUCTURAL MUST
( sn $ cn )
  MAY ( userPassword $ telephoneNumber $ seeAlso $ description
)
  X-ORIGIN 'RFC 4519' )
```

  Entries all have an attribute identifying their object classes(es), called `objectClass`.

  Object class definitions start with an object identifier (OID), and a short name that is easier to remember.

  The definition here says that the person object class inherits from the `top` object class, which is the top-level parent of all object classes.

  An entry can have one STRUCTURAL object class inheritance branch, such as `top` → `person` → `organizationalPerson` → `inetOrgPerson`. Entries can have multiple AUXILIARY object classes. The object class defines the attribute types that must and may be present on entries of the object class.

- An *attribute syntax* constrains what directory clients can store as attribute values.

  An attribute syntax is identified in an attribute type definition by its OID. String-based syntax OIDs are optionally followed by a number set between braces. The number represents a minimum upper bound on the number of characters in the attribute value. For example, in the attribute type definition shown above, the syntax is `1.3.6.1.4.1.1466.115.121.1.26{256}` , IA5 string. An IA5 string (composed of characters from the international version of the ASCII character set) can contain at least 256 characters.

  You can find a table matching attribute syntax OIDs with their human-readable names in RFC 4517, Appendix A. Summary of Syntax Object Identifiers. The RFC describes attribute syntaxes in detail. You can list attribute syntaxes with the **`dsconfig`** command.

  If you are trying unsuccessfully to import non-compliant data, clean the data before importing it. If cleaning the data is not an option, read Import Legacy Data.

  When creating attribute type definitions, use existing attribute syntaxes where possible. If you must create your own attribute syntax, then consider the schema extensions in Update LDAP Schema.

  Although attribute syntaxes are often specified in attribute type definitions, DS servers do not always check that attribute values comply with attribute syntaxes. DS servers do enforce compliance by default for the following to avoid bad directory data:

  - Certificates
  - Country strings
  - Directory strings
  - JPEG photos
  - Telephone numbers

- *Matching rules* define how to compare attribute values to assertion values for LDAP search and LDAP compare operations.

  For example, suppose you search with the filter `(uid=bjensen)` . The assertion value in this case is `bjensen` .

  DS servers have the following schema definition for the user ID attribute:

  ```
  attributeTypes: ( 0.9.2342.19200300.100.1.1 NAME ( 'uid'
  'userid' )
   EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256} X-ORIGIN 'RFC 4519'
  )
  ```

When finding an equality match for your search, servers use the `caseIgnoreMatch` matching rule to check for user ID attribute values that equal `bjensen` .

You can read the schema definitions for matching rules that the server supports by performing an LDAP search:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --baseDn cn=schema \
 --searchScope base \
 "(&)" \
 matchingRules
```

Notice that many matching rules support string collation in languages other than English. For the full list of string collation matching rules, see Supported Locales. For the list of other matching rules, see Matching Rules.

Matching rules enable directory clients to compare other values besides strings.

DS servers expose schema over protocol through the `cn=schema` entry. The server stores the schema definitions in LDIF format files in the `db/schema/` directory. When you set up a server, the process copies many definitions to this location.

Update LDAP Schema

DS servers allow you to update directory schema definitions while the server is running. You can add support for new types of attributes and entries without interrupting the directory service. DS servers replicate schema definitions, propagating them to other replicas automatically.

You update schema either by:

- Performing LDAP modify operations while the server is running.
- Adding schema files to the `db/schema/` directory before starting the server.

Before adding schema definitions, take note of the following points:

- Define DS schema elements in LDIF.

  For examples, see the build in schema files in the `db/schema/` directory.

- Add your schema definitions in a file prefixed with a higher number than built-in files, such as `99-user.ldif`, which is the default filename when you modify schema over LDAP.

  On startup, the DS server reads schema files in order, sorted alphanumerically. Your definitions likely depend on others that the server should read first.

- Define a schema element before referencing it in other definitions.

  For example, make sure you define an attribute type before using it in an object class definition.

For an example, see Custom Schema.

DS servers support the standard LDAP schema definitions described in RFC 4512, section 4.1. They also support the following extensions:

Schema Extensions for All Types

X-DEPRECATED-SINCE

This specifies a release that deprecates the schema element.

Example: `X-DEPRECATED-SINCE: version`

X-ORIGIN

This specifies the origin of a schema element.

Examples:

- `X-ORIGIN 'RFC 4519'`
- `X-ORIGIN 'draft-ietf-ldup-subentry'`
- `X-ORIGIN 'DS Directory Server'`

X-SCHEMA-FILE

This specifies the relative path to the schema file containing the schema element.

Schema definitions are located in `/path/to/opendj/db/schema/*.ldif` files.

Example: `X-SCHEMA-FILE '00-core.ldif'`.

X-STABILITY

Used to specify the interface stability of the schema element.

This extension takes one of the following values:

- `Evolving`
- `Internal`
- `Removed`
- `Stable`
- `Technology Preview`

Schema Extensions for Syntaxes

Extensions to syntax definitions requires additional code to support syntax checking. DS servers support the following extensions for their particular use cases:

X-ENUM

This defines a syntax that is an enumeration of values.

The following attribute syntax description defines a syntax allowing four possible attribute values:

```
ldapSyntaxes: ( security-label-syntax-oid DESC 'Security Label'
 X-ENUM ( 'top-secret' 'secret' 'confidential' 'unclassified' )
)
```

X-PATTERN

This defines a syntax based on a regular expression pattern. Valid regular expressions are those defined for java.util.regex.Pattern.

The following attribute syntax description defines a simple, lenient SIP phone URI syntax check:

```
ldapSyntaxes: ( simple-sip-uri-syntax-oid DESC 'Lenient SIP URI
Syntax'
  X-PATTERN '^sip:[a-zA-Z0-9.]+@[a-zA-Z0-9.]+(:[0-9]+)?$' )
```

X-SUBST

This specifies a substitute syntax to use for one that DS servers do not implement.

The following example substitutes Directory String syntax, OID `1.3.6.1.4.1.1466.115.121.1.15`, for a syntax that DS servers do not implement:

```
ldapSyntaxes: ( non-implemented-syntax-oid DESC 'Not
Implemented in DS'
  X-SUBST '1.3.6.1.4.1.1466.115.121.1.15' )
```

Schema Extensions for Attributes

X-APPROX

`X-APPROX` specifies a non-default approximate matching rule for an attribute type.

The default is the double metaphone approximate match.

Custom Schema

This example updates the LDAP schema while the server is online. It defines a custom enumeration syntax using the attribute type and a custom object class that uses the

attribute:

- A custom enumeration syntax using the `X-ENUM` extension.
- A custom attribute type using the custom syntax.
- A custom object class for entries that have the custom attribute:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password <<EOF
dn: cn=schema
changetype: modify
add: ldapSyntaxes
ldapSyntaxes: ( temporary-syntax-oid
  DESC 'Custom enumeration syntax'
  X-ENUM ( 'bronze' 'silver' 'gold' )
  X-ORIGIN 'DS Documentation Examples'
  X-SCHEMA-FILE '99-user.ldif' )
-
add: attributeTypes
attributeTypes: ( temporary-attr-oid
  NAME 'myEnum'
  DESC 'Custom attribute type for'
  SYNTAX temporary-syntax-oid
  USAGE userApplications
  X-ORIGIN 'DS Documentation Examples'
  X-SCHEMA-FILE '99-user.ldif' )
-
add: objectClasses
objectClasses: ( temporary-oc-oid
  NAME 'myEnumObjectClass'
  DESC 'Custom object class for entries with a myEnum attribute'
  SUP top
  AUXILIARY
  MAY myEnum
  X-ORIGIN 'DS Documentation Examples'
  X-SCHEMA-FILE '99-user.ldif' )

EOF
```

```
# MODIFY operation successful for DN cn=schema
```

Notice the follow properties of this update to the schema definitions:

- The `ldapSyntaxes` definition comes before the `attributeTypes` definition that uses the syntax.

  The `attributeTypes` definition comes before the `objectClasses` definition that uses the attribute type.

- Each definition has a temporary OID of the form `temporary-*-oid`.

  While you develop new schema definitions, temporary OIDs are fine. Get permanent, correctly assigned OIDs before using schema definitions in production.

- Each definition has a `DESC` (description) string intended for human readers.
- Each definition specifies its origin with the extension `X-ORIGIN 'DS Documentation Examples'`.
- Each definition specifies its schema file with the extension `X-SCHEMA-FILE '99-user.ldif'`.
- The syntax definition has no name, as it is referenced internally by OID only.
- `X-ENUM ( 'bronze' 'silver' 'gold' )` indicates that the syntax allows three values, `bronze`, `silver`, `gold`.

  DS servers reject other values for attributes with this syntax.

- The attribute type named `myEnum` has these properties:
  - It uses the enumeration syntax, `SYNTAX temporary-syntax-oid`.
  - It can only have one value at a time, `SINGLE-VALUE`.

    The default, if you omit `SINGLE-VALUE`, is to allow multiple values.
  - It is intended for use by user applications, `USAGE userApplications`.
- The object class named `myEnumObjectClass` has these properties:
  - Its parent for inheritance is the top-level object class, `SUP top`.

    `top` is the abstract parent of all structural object class hierarchies, so all object classes inherit from it.
  - It is an auxiliary object class, `AUXILIARY`.

    Auxiliary object classes are used to augment attributes of entries that already have a structural object class.
  - It defines no required attributes (no `MUST`).
  - It defines one optional attribute which is the custom attribute, `MAY myEnum`.

After adding the schema definitions, you can add the attribute to an entry as shown in the following example:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password <<EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: myEnumObjectClass
-
add: myEnum
myEnum: silver

EOF

# MODIFY operation successful for DN
uid=bjensen,ou=People,dc=example,dc=com
```

As shown in the following example, the attribute syntax prevents users from setting an attribute value that is not specified in your enumeration:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password <<EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: myEnum
myEnum: wrong value

EOF

# The LDAP modify request failed: 21 (Invalid Attribute Syntax)
# Additional Information:  When attempting to modify entry
```

```
uid=bjensen,ou=People,dc=example,dc=com to replace the set of
values for attribute myEnum, value "wrong value" was found to be
invalid according to the associated syntax: The provided value
"wrong value" cannot be parsed because it is not allowed by
enumeration syntax with OID "temporary-syntax-oid"
```

For more examples, read the built-in schema definition files in the `db/schema/` directory.

Schema and JSON

DS software has the following features for working with JSON objects:

- **RESTful HTTP access to directory services**

  If you have LDAP data, but HTTP client applications want JSON over HTTP instead, DS software can expose your LDAP data as JSON resources over HTTP to REST clients. You can configure how LDAP entries map to JSON resources.

  There is no requirement to change LDAP schema definitions before using this feature. To get started, see DS REST APIs.

- **JSON syntax LDAP attributes**

  If you have LDAP client applications that store JSON in the directory, you can define LDAP attributes that have `Json` syntax.

  The following schema excerpt defines an attribute called `json` with case-insensitive matching:

  ```
  attributeTypes: ( json-attribute-oid NAME 'json'
   SYNTAX 1.3.6.1.4.1.36733.2.1.3.1 EQUALITY
  caseIgnoreJsonQueryMatch
   SINGLE-VALUE X-ORIGIN 'DS Documentation Examples' )
  ```

  Notice that the JSON syntax OID is `1.3.6.1.4.1.36733.2.1.3.1`. The definition above uses the (default) `caseIgnoreJsonQueryMatch` matching rule for equality. As explained later in this page, you might want to choose different matching rules for your JSON attributes.

  When DS servers receive update requests for `Json` syntax attributes, they expect valid JSON objects. By default, `Json` syntax attribute values must comply with *The JavaScript Object Notation (JSON) Data Interchange Format*, described in RFC 7159. You can use the advanced core schema configuration option `json-validation-policy` to have the server be more lenient in what it accepts, or to disable JSON syntax checking.

- **Configurable indexing for JSON attributes**

  When you store JSON attributes in the directory, you can index every field in each JSON attribute value, or you can index only what you use.

  As for other LDAP attributes, the indexes depend on the matching rule defined for the JSON syntax attribute.

DS servers treat JSON syntax attribute values as objects. Two JSON values may be considered equivalent despite differences in their string representations. The following JSON objects can be considered equivalent because each field has the same value. Their string representations are different, however:

```
[
 { "id": "bjensen", "given-name": "Barbara", "surname": "Jensen"
},
 {"surname":"Jensen","given-name":"Barbara","id":"bjensen"}
]
```

Unlike other objects with their own LDAP attribute syntaxes, such as X.509 certificates, two JSON objects with completely different structures (different field names and types) are still both JSON. Nothing in the JSON syntax alone tells the server anything about what a JSON object must and may contain.

When defining LDAP schema for JSON attributes, it helps therefore to understand the structure of the expected JSON. Will the attribute values be arbitrary JSON objects, or JSON objects whose structure is governed by some common schema? If a JSON attribute value is an arbitrary object, you can do little to optimize how it is indexed or compared. If the value is a structured object, however, you can configure optimizations based on the structure.

For structured JSON objects, the definition of JSON object equality is what enables you to pick the optimal matching rule for the LDAP schema definition. The matching rule determines how the server indexes the attribute, and how the server compares two JSON values for equality. You can define equality in the following ways:

- Two JSON objects are equal if *all* fields have the same values.

  By this definition, `{"a": 1, "b": true}` equals `{"b":true,"a":1}`. However, `{"a": 1, "b": true} and {"a": 1, "b": "true"}` are different.

- Two JSON objects ` equal if *some* fields have the same values. Other fields are ignored when comparing for equality.

  For example, take the case where two JSON objects are considered equal if they have the same "_id" values. By this definition, `{"_id":1,"b":true}` equals `{"_id":1,"b":false}`. However, `{"_id":1,"b":true}` and `{"_id":2,"b":true}` are different.

DS servers have built-in matching rules for the case where equality means "_id" values are equal. If the fields to compare are different from "_id", you must define your own matching rule and configure a custom schema provider that implements it. This following table helps you choose a JSON equality matching rule:

JSON Content	Two JSON Objects are Equal if...	Use One of These Matching Rules
Arbitrary (any valid JSON is allowed)	All fields have the same values.	`caseExactJsonQueryMatch` `caseIgnoreJsonQueryMatch`
Structured	All fields have the same values.	`caseExactJsonQueryMatch` `caseIgnoreJsonQueryMatch`
Structured	"_id" fields have the same values. Additional fields are ignored when comparing for equality.	`caseExactJsonIdMatch` `caseIgnoreJsonIdMatch`
Structured	One or more other fields have the same values. Additional fields are ignored when comparing for equality.	When using this matching rule, create a custom `json-equality-matching-rule` Schema Provider. The custom schema provider must include all the necessary properties and reference the custom field(s). See JSON Equality Matching Rule Index.

When you choose an equality matching rule in the LDAP attribute definition, you are also choosing the default that applies in an LDAP search filter equality assertion. For example, `caseIgnoreJsonQueryMatch` works with filters such as `"(json=id eq 'bjensen')"`. `caseIgnoreJsonIdMatch` works with filters such as `'(json={"_id":"bjensen"})'`.

DS servers also implement JSON ordering matching rules for determining the relative order of two JSON values using a custom set of rules. You can select which JSON fields should be used for performing the ordering match. You can also define whether those

fields that contain strings should be normalized before comparison by trimming white space or ignoring case differences. DS servers can implement JSON ordering matching rules on demand when presented with an extended server-side sort request, as described in Server-Side Sort. If, however, you define them statically in your LDAP schema, then you must implement them by creating a custom `json-ordering-matching-rule` Schema Provider. For details about the `json-ordering-matching-rule` object's properties, see JSON Ordering Matching Rule.

For examples showing how to add LDAP schema for new attributes, see Update LDAP Schema. For examples showing how to index JSON attributes, see Indexes for JSON.

Import Legacy Data

By default, DS servers accept data that follows the schema for allowable and rejected data. You might have legacy data from a directory service that is more lenient, allowing non-standard constructions such as multiple structural object classes per entry, not checking attribute value syntax, or even not respecting schema definitions.

For example, when importing data with multiple structural object classes defined per entry, you can relax schema checking to warn rather than reject entries having this issue:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --set single-structural-objectclass-behavior:warn \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

You can allow attribute values that do not respect the defined syntax:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --set invalid-attribute-syntax-behavior:warn \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

```
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

You can even turn off schema checking altogether. Only turn off schema checking *when you are absolutely sure that the entries and attributes already respect the schema definitions*:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --set check-schema:false \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Turning off schema checking can potentially boost import performance.

Standard Schema

DS servers provide many standard schema definitions. For full descriptions, see the Schema Reference. Find the definitions in LDIF files in the `/path/to/opendj/db/schema/` directory:

File	Description
`00-core.ldif`	Schema definitions for the following Internet-Drafts, RFCs, and standards: • draft-boreham-numsubordinates • draft-findlay-ldap-groupofentries • draft-good-ldap-changelog • draft-howard-namedobject • draft-ietf-ldup-subentry • draft-wahl-ldap-adminaddr • RFC 1274 • RFC 2079 • RFC 2256 • RFC 2798 • RFC 3045 • RFC 3296 • RFC 3671 • RFC 3672 • RFC 4512 • RFC 4519 • RFC 4523 • RFC 4524 • RFC 4530 • RFC 5020 • X.501
`01-pwpolicy.ldif`	Schema for draft-behera-ldap-password-policy (Draft 09), which defines a mechanism for storing password policy information in an LDAP directory server.
`02-config.ldif`	This file contains the attribute type and objectclass definitions for use with the server configuration.
`03-changelog.ldif`	Schema for draft-good-ldap-changelog, which defines a mechanism for storing information about changes to directory server data.

File	Description
`03-rfc2713.ldif`	Schema for RFC 2713, which defines a mechanism for storing serialized Java objects in the directory server.
`03-rfc2714.ldif`	Schema for RFC 2714, which defines a mechanism for storing CORBA objects in the directory server.
`03-rfc2739.ldif`	Schema for RFC 2739, which defines a mechanism for storing calendar and vCard objects in the directory server. Be aware that the definition in RFC 2739 contains a number of errors. This schema file has been altered from the standard definition to fix a number of those problems.
`03-rfc2926.ldif`	Schema for RFC 2926, which defines a mechanism for mapping between Service Location Protocol (SLP) advertisements and LDAP.
`03-rfc3112.ldif`	Schema for RFC 3112, which defines the authentication password schema.
`03-rfc3712.ldif`	Schema for RFC 3712, which defines a mechanism for storing printer information in the directory server.
`03-uddiv3.ldif`	Schema for RFC 4403, which defines a mechanism for storing UDDIv3 information in the directory server.
`04-rfc2307bis.ldif`	Schema for draft-howard-rfc2307bis, which defines a mechanism for storing naming service information in the directory server.
`05-rfc4876.ldif`	Schema for RFC 4876, which defines a schema for storing Directory User Agent (DUA) profiles and preferences in the directory server.
`05-samba.ldif`	Schema required when storing Samba user accounts in the directory server.
`05-solaris.ldif`	Schema required for Solaris and OpenSolaris LDAP naming services.
`06-compat.ldif`	Backwards-compatible schema for use in the server configuration.

Indexes

About Indexes

A basic, standard directory feature is the ability to respond quickly to searches.

An LDAP search specifies the information that directly affects how long the directory might take to respond:

- The base DN for the search.

  The more specific the base DN, the less information to check during the search. For example, a request with base DN `dc=example,dc=com` potentially involves checking many more entries than a request with base DN `uid=bjensen,ou=people,dc=example,dc=com`.

- The scope of the search.

  A subtree or one-level scope targets many entries, whereas a base search is limited to one entry.

- The search filter to match.

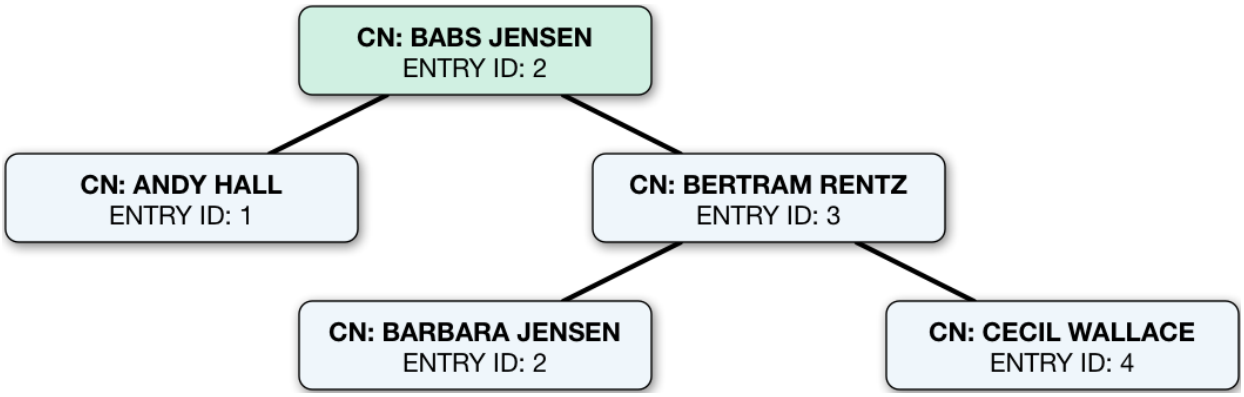
  A search filter asserts that for an entry to match, it has an attribute that corresponds to some value. For example, `(cn=Babs Jensen)` asserts that `cn` must have a value that equals `Babs Jensen`.

  A directory server would waste resources checking all entries for a match. Instead, directory servers maintain indexes to expedite checking for a match.

LDAP directory servers disallow searches that cannot be handled expediently using indexes. Maintaining appropriate indexes is a key aspect of directory administration.

The role of an index is to answer the question, "Which entries have an attribute with this corresponding value?" Each index is therefore specific to an attribute. Each index is also specific to the comparison implied in the search filter. For example, a directory server maintains distinct indexes for exact (equality) matching and for substring matching. The types of indexes are explained in Index Types. Furthermore, indexes are configured in specific directory backends.

An index is implemented as a tree of key-value pairs. The key is a form of the value to match, such as `babs jensen`. The value is a list of IDs for entries that match the key. The figure that follows shows an equality (case ignore exact match) index with five keys from a total of four entries. If the data set were large, there could be more than one entry ID per key:



This is how DS directory servers use indexes. When the search filter is `(cn=Babs Jensen)`, the directory server retrieves the IDs for entries with a CN matching `Babs Jensen` from the equality index of the CN attribute. (For a complex filter, it might optimize the search by changing the order in which it uses the indexes.) A successful result is zero or more entry IDs. These are the candidate result entries.

For each candidate, the DS directory server retrieves the entry by ID from a special system index called `id2entry`. As its name suggests, this index returns an entry for an entry ID. If there is a match, and the client application has the right to access to the data, the directory server returns the search result. It continues this process until no candidates are left.

If there are no indexes that correspond to a search request, the server must check for a match against every entry in the scope of the search. Evaluating every entry for a match is referred to as an *unindexed* search. An unindexed search is an expensive operation, particularly for large directories. A server refuses unindexed searches unless the user has specific permission to make such requests. The permission to perform an unindexed search is granted with the `unindexed-search` privilege. This privilege is reserved for the directory superuser by default. It should not be granted lightly.

If the number of entries is smaller than the default resource limits, you can still perform what appear to be unindexed searches, meaning searches with filters for which no index appears to exist. That is because the `dn2id` index returns all user data entries without hitting a resource limit that would make the search unindexed.

Use cases that may call for unindexed searches include the following:

- An application must periodically retrieve a very large amount of directory data all at once through an LDAP search.

  For example, an application performs an LDAP search to retrieve everything in the directory once a week as part of a batch job that runs during off hours.

  Make sure the application has no resource limits. For details, see Resource Limits.
- A directory data administrator occasionally browses directory data through a graphical UI without initially knowing what they are looking for or how to narrow the search.

For this case, DS directory servers can sort unindexed search results as long as they are paged.

This capability has the following limitations:

- The simple paged results control must specify a page size that is less than or equal to the `index-entry-limit` (default: 4000).
- For each page, the server reads the entire backend database, retaining page size number of sorted entries.

Alternatively, DS directory servers can use an appropriately configured VLV index to sort results for an unindexed search. For details, see VLV for Paged Server-Side Sort.

What To Index

DS directory server search performance depends on indexes. The default settings are fine for evaluating DS software, and they work well with sample data. The default settings do not necessarily fit your directory data, and the searches your applications perform.

Necessary Indexes

Index maintenance has its costs. Every time an indexed attribute is updated, the server must update each affected index to reflect the change. This is wasteful if the index is not used. Indexes, especially substring indexes, can occupy more memory and disk space than the corresponding data.

Aim to maintain only indexes that speed up appropriate searches, and that allow the server to operate properly. The former indexes depend on how directory users search, and require thought and investigation. The latter includes non-configurable internal indexes, that should not change.

Begin by reviewing the attributes of your directory data. Which attributes would you expect to see in a search filter? If an attribute is going to show up frequently in reasonable search filters, then index it.

Compare your guesses with what you see actually happening in the directory. One approach is to review the access log for search results with additional items like `unindexed` . The following example shows the relevant fields in an access log message:

```
{
  "request": {
    "protocol": "LDAP",
    "operation": "SEARCH"
  },
  "response": {
```

```
    "detail": "You do not have sufficient privileges to perform an
unindexed search",
    "additionalItems": {
      "unindexed": null
    }
  }
}
```

Review the messages in the access log, as they also specify the search filter and scope. Understand the search that led to each unindexed search. If the filter is appropriate and frequently used, add an index to facilitate the search. You can either consume the access logs to determine how often a search filter is used, or monitor what is happening in the directory with the index analysis feature.

DS servers provide the index analysis feature to collect information about filters in search requests. You can activate the index analysis mechanism using the **`dsconfig set-backend-prop`** command:

```
$ dsconfig \
 set-backend-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --set index-filter-analyzer-enabled:true \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
```

The command causes the server to analyze filters used, and to keep the results in memory. You can read the results as monitoring information:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN ds-cfg-backend-id=dsEvaluation,cn=Backends,cn=monitor \
 --searchScope base \
 "(&)" \
 ds-mon-backend-filter-use-start-time ds-mon-backend-filter-use-
```

```
indexed ds-mon-backend-filter-use-unindexed ds-mon-backend-filter-
use

dn: ds-cfg-backend-id=dsEvaluation,cn=backends,cn=monitor
ds-mon-backend-filter-use-start-time: <timestamp>
ds-mon-backend-filter-use-indexed: 2
ds-mon-backend-filter-use-unindexed: 3
ds-mon-backend-filter-use: {"search-filter":"
(employeenumber=86182)","nb-hits":1,"latest-failure-
reason":"caseIgnoreMatch index type is disabled for the
employeeNumber attribute"}
```

The `ds-mon-backend-filter-use` values include the following fields:

search-filter
: The LDAP search filter.

nb-hits
: The number of times the filter was used.

latest-failure-reason
: A message describing why the server could not use any index for this filter.

The output can include filters for internal use, such as `(aci=*)`. In the example above, you see a filter used by a client application.

In the example, a search filter that led to an unindexed search, `(employeenumber=86182)`, had no matches because, "caseIgnoreMatch index type is disabled for the employeeNumber attribute". Some client application has tried to find users by employee number, but no index exists for that purpose. If this appears regularly as a frequent search, add an employee number index.

To avoid impacting server performance, turn off index analysis after you collect the information you need. Turn off index analysis with the **`dsconfig set-backend-prop`** command:

```
$ dsconfig \
 set-backend-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --set index-filter-analyzer-enabled:false \
 --no-prompt \
```

```
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
```

Directory users might complain to you that their searches are refused because they are unindexed. Ask for the result code, additional information, and search filter. DS directory servers respond to LDAP client applications that attempt unindexed searches with a result code of 50 and additional information about the unindexed search. The following example attempts, anonymously, to get the entries for all users whose email address ends in `.com`:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN ou=people,dc=example,dc=com \
 "(&(mail=*.com)(objectclass=person))"

# The LDAP search request failed: 50 (Insufficient Access Rights)
# Additional Information:  You do not have sufficient privileges
to perform an unindexed search
```

Before you change settings to permit a search, understand why the user wants to perform an unindexed search.

Perhaps they are unintentionally requesting an unindexed search. If so, you can help them find a less expensive search, by using an approach that limits the number of candidate result entries. For example, if a GUI application lets a user browse a group of entries, the application could use a browsing index to retrieve a block of entries for each screen, rather than retrieving all the entries at once.

Perhaps they do have a legitimate reason to get the full list of all entries in one operation, such as regularly rebuilding some database that depends on the directory. If so, assign their application's account the `unindexed-search` privilege.

In addition to responding to client search requests, a server performs internal searches. Internal searches let the server retrieve data needed for a request, and maintain internal state information. In some cases internal searches can become unindexed. When this happens, the server logs a warning similar to the following:

```
The server is performing an unindexed internal search request
with base DN '%s', scope '%s', and filter '%s'. Unindexed internal
searches are usually unexpected and could impact performance.
```

```
Please verify that that backend's indexes are configured correctly
for these search parameters.
```

When you see a message like this in the server log, take these actions:

- Figure out which indexes are missing, and add them.

  For details, see Debug Search Indexes, and Configure Indexes.

- Check the integrity of the indexes.

  For details, see Verify Indexes.

- If the relevant indexes exist, and you have verified that they are sound, the index entry limit might be too low.

  This can happen, for example, in directory servers with more than 4000 groups in a single backend. For details, see Index Entry Limits.

- If you have made the changes described in the steps above, and problem persists, contact technical support.

Debug Search Indexes

Sometimes it is not obvious by inspection how a directory server processes a given search request. The directory superuser can gain insight with the `debugsearchindex` attribute.

The default global access control prevents users from reading the `debugsearchindex` attribute. To allow an administrator to read the attribute, add a global ACI such as the following:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --add global-aci:"(targetattr=\"debugsearchindex\")(version 3.0;
acl \"Debug search indexes\"; \
  allow (read,search,compare)
userdn=\"ldap:///uid=user.0,ou=people,dc=example,dc=com\";)" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

NOTE

NOTE

The format of `debugsearchindex` values has interface stability: Internal.

The values are intended to be read by human beings, not scripts. If you do write scripts that interpret `debugsearchindex` values, be aware that they are not stable. Be prepared to adapt your scripts for every upgrade or patch.

The `debugsearchindex` attribute value indicates how the server would process the search. The server use its indexes to prepare a set of candidate entries. It iterates through the set to compare candidates with the search filter, returning entries that match. The following example demonstrates this feature for a subtree search with a complex filter:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=user.0,ou=people,dc=example,dc=com \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(&(objectclass=person)(givenName=aa*))" \
 debugsearchindex | sed -n -e "s/^debugsearchindex: //p"

{
  "filter": {
    "intersection": [
      {
        "index": "givenName.caseIgnoreSubstringsMatch:6",
        "range": "[aa,ab[",
        "candidates": 171,
        "retained": 171
      },
      {
        "index": "givenName.caseIgnoreMatch",
        "range": "[aa,ab[",
        "candidates": 50,
        "retained": 50
      },
      {
        "filter": "(objectclass=person)",
        "union": [
          {
            "index": "objectClass.objectIdentifierMatch",
```

```
            "exact": "person",
            "candidates": "[LIMIT-EXCEEDED]"
          }
        ],
        "candidates": "[LIMIT-EXCEEDED]",
        "retained": 50
      }
    ],
    "candidates": 50
  },
  "scope": {
    "type": "sub",
    "candidates": "[NOT-INDEXED]",
    "retained": 50
  },
  "final": 50
}
```

The filter in the example matches person entries whose given name starts with `aa`. The search scope is not explicitly specified, so the scope defaults to the subtree including the base DN.

Notice that the `debugsearchindex` value has the following top-level fields:

- (Optional) `"vlv"` describes how the server uses VLV indexes.

  The VLV field is not applicable for this example, and so is not present.
- `"filter"` describes how the server uses the search filter to narrow the set of candidates.
- `"scope"` describes how the server uses the search scope.
- `"final"` indicates the final number of candidates in the set.

In the output, notice that the server uses the equality and substring indexes to find candidate entries whose given name starts with `aa`. If the filter indicated given names *containing* aa, as in `givenName=**aa**`, the server would rely only on the substring index.

Notice that the output for the `(objectclass=person)` portion of the filter shows `"candidates": "[LIMIT-EXCEEDED]"`. In this case, there are so many entries matching the value specified that the index is not useful for narrowing the set of candidates. The scope is also not useful for narrowing the set of candidates. Ultimately, however, the `givenName` indexes help the server to narrow the set of candidates. The overall search is indexed and the result is 50 matching entries.

The following example shows a subtree search for accounts with initials starting either with `aa` or with `zz`:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 --bindDN uid=user.0,ou=people,dc=example,dc=com \
 --bindPassword password \
 "(|(initials=aa*)(initials=zz*))" \
 debugsearchindex | sed -n -e "s/^debugsearchindex: //p"

{
  "filter": {
    "union": [
      {
        "filter": "(initials=aa*)",
        "index": "initials.presence",
        "diagnostic": "not indexed",
        "candidates": "[NOT-INDEXED]"
      }
    ],
    "candidates": "[NOT-INDEXED]"
  },
  "scope": {
    "type": "sub",
    "candidates": "[NOT-INDEXED]",
    "retained": "[NOT-INDEXED]"
  },
  "final": "[NOT-INDEXED]"
}
```

As shown in the output, the search is not indexed. To fix this, index the `initials` attribute:

```
$ dsconfig \
 create-backend-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name initials \
 --set index-type:equality \
```

```
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ rebuild-index \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --baseDn dc=example,dc=com \
 --index initials \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
```

After configuring and building the new index, try the same search again:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 --bindDN uid=user.0,ou=people,dc=example,dc=com \
 --bindPassword password \
 "(|(initials=aa*)(initials=zz*))" \
 debugsearchindex | sed -n -e "s/^debugsearchindex: //p"

{
  "filter": {
    "union": [
      {
        "filter": "(initials=aa*)",
        "intersection": [
          {
            "index": "initials.caseIgnoreSubstringsMatch:6",
            "range": "[aa,ab[",
            "diagnostic": "not indexed",
            "tryPresenceIndex": {
              "index": "initials.presence",
              "diagnostic": "not indexed"
            },
            "candidates": "[NOT-INDEXED]",
            "retained": "[NOT-INDEXED]"
          },
```

```
          {
            "index": "initials.caseIgnoreMatch",
            "range": "[aa,ab[",
            "candidates": 378,
            "retained": 378
          }
        ],
        "candidates": 378
      },
      {
        "filter": "(initials=zz*)",
        "intersection": [
          {
            "index": "initials.caseIgnoreSubstringsMatch:6",
            "range": "[zz,z{[",
            "diagnostic": "not indexed",
            "tryPresenceIndex": {
              "index": "initials.presence",
              "diagnostic": "not indexed"
            },
            "candidates": "[NOT-INDEXED]",
            "retained": "[NOT-INDEXED]"
          },
          {
            "index": "initials.caseIgnoreMatch",
            "range": "[zz,z{[",
            "candidates": 26,
            "retained": 26
          }
        ],
        "candidates": 26
      }
    ],
    "candidates": 404
  },
  "scope": {
    "type": "sub",
    "candidates": "[NOT-INDEXED]",
    "retained": 404
  },
  "final": 404
}
```


Notice that the server can narrow the list of candidates using the equality index you created. The server would require a substring index instead of an equality index if the

filter were not matching initial strings.

If an index already exists, but you suspect it is not working properly, see Verify Indexes.

Index Types

DS directory servers support multiple index types, each corresponding to a different type of search.

View what is indexed by using the **`backendstat list-indexes`** command. For details about a particular index, you can use the **`backendstat dump-index`** command.

Presence Index

A presence index matches an attribute that is present on the entry, regardless of the value. By default, the `aci` attribute is indexed for presence:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(aci=*)" \
 aci
```

A presence index takes up less space than other indexes. In a presence index, there is just one key with a list of IDs.

The following command examines the ACI presence index for a server configured with the evaluation profile:

```
$ stop-ds

$ backendstat \
 dump-index \
 --backendId dsEvaluation \
 --baseDn dc=example,dc=com \
 --indexName aci.presence

Key (len 1): PRESENCE
Value (len 3): [COUNT:2] 1 9
```

```
Total Records: 1
Total / Average Key Size: 1 bytes / 1 bytes
Total / Average Data Size: 3 bytes / 3 bytes
```

In this case, entries with ACI attributes have IDs `1` and `9`.

Equality Index

An equality index matches values that correspond exactly (generally ignoring case) to those in search filters. An equality index requires clients to match values without wildcards or misspellings:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 mail

dn: uid=bjensen,ou=People,dc=example,dc=com
mail: bjensen@example.com
```

An equality index has one list of entry IDs for each attribute value. Depending on the backend implementation, the keys in a case-insensitive index might not be strings. For example, a key of `6A656E73656E` could represent `jensen`.

The following command examines the SN equality index for a server configured with the evaluation profile:

```
$ stop-ds

$ backendstat \
 dump-index \
 --backendID dsEvaluation \
 --baseDN dc=example,dc=com \
 --indexName sn.caseIgnoreMatch | grep -A 1 "jensen$"

Key (len 6): jensen
Value (len 26): [COUNT:17] 18 31 32 66 79 94 133 134 150 5996
19415 32834 46253 59672 73091 86510 99929
```

In this case, there are 17 entries that have an SN of Jensen.

Unless the keys are encrypted, the server can reuse an equality index for ordering and initial substring searches.

Approximate Index

An approximate index matches values that "sound like" those provided in the filter. An approximate index on `sn` lets client applications find people even when they misspell surnames:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(&(sn~=Jansen)(cn=Babs*))" \
 cn

dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
```

An approximate index squashes attribute values into a normalized form.

The following command examines an SN approximate index added to a server configured with the evaluation profile:

```
$ stop-ds

$ backendstat \
 dump-index \
 --backendID dsEvaluation \
 --baseDN dc=example,dc=com \
 --indexName sn.ds-mr-double-metaphone-approx | grep -A 1 "JNSN$"

Key (len 4): JNSN
Value (len 83): [COUNT:74] 18 31 32 59 66 79 94 133 134 150 5928
5939 5940 5941 5996 5997 6033 6034 19347 19358 19359 19360 19415
19416 19452 19453 32766 32777 32778 32779 32834 32835 32871 32872
46185 46196 46197 46198 46253 46254 46290 46291 59604 59615 59616
59617 59672 59673 59709 59710 73023 73034 73035 73036 73091 73092
```

```
73128 73129 86442 86453 86454 86455 86510 86511 86547 86548 99861
99872 99873 99874 99929 99930 99966 99967
```

In this case, there are 74 entries that have an SN that sounds like Jensen.

Substring Index

A substring index matches values that are specified with wildcards in the filter. Substring indexes can be expensive to maintain, especially for large attribute values:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(cn=Barb*)" \
 cn

dn: uid=bfrancis,ou=People,dc=example,dc=com
cn: Barbara Francis

dn: uid=bhal2,ou=People,dc=example,dc=com
cn: Barbara Hall

dn: uid=bjablons,ou=People,dc=example,dc=com
cn: Barbara Jablonski

dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen

dn: uid=bmaddox,ou=People,dc=example,dc=com
cn: Barbara Maddox
```

In a substring index, there are enough keys to match any substring in the attribute values. Each key is associated with a list of IDs. The default maximum size of a substring key is 6 bytes.

The following command examines an SN substring index for a server configured with the evaluation profile:

```
$ stop-ds

$ backendstat \
dump-index \
 --backendID dsEvaluation \
 --baseDN dc=example,dc=com \
 --indexName sn.caseIgnoreSubstringsMatch:6

...
Key (len 1): e
Value (len 25): [COUNT:22] ...
...
Key (len 2): en
Value (len 15): [COUNT:12] ...
...
Key (len 3): ens
Value (len 3): [COUNT:1] 147
Key (len 5): ensen
Value (len 10): [COUNT:9] 18 31 32 66 79 94 133 134 150
...
Key (len 6): jensen
Value (len 10): [COUNT:9] 18 31 32 66 79 94 133 134 150
...
Key (len 1): n
Value (len 35): [COUNT:32] ...
...
Key (len 2): ns
Value (len 3): [COUNT:1] 147
Key (len 4): nsen
Value (len 10): [COUNT:9] 18 31 32 66 79 94 133 134 150
...
Key (len 1): s
Value (len 13): [COUNT:12] 12 26 47 64 95 98 108 131 135 147 149
154
...
Key (len 2): se
Value (len 7): [COUNT:6] 52 58 75 117 123 148
Key (len 3): sen
Value (len 10): [COUNT:9] 18 31 32 66 79 94 133 134 150
...
```

In this case, the SN value Jensen shares substrings with many other entries. The size of the lists and number of keys make a substring index much more expensive to maintain than other indexes. This is particularly true for longer attribute values.

Ordering Index

An ordering index is used to match values for a filter that specifies a range. For example, the `ds-sync-hist` attribute used by replication has an ordering index by default. Searches on that attribute often seek entries with changes more recent than the last time a search was performed.

The following example shows a search that specifies a range on the SN attribute value:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(sn>=zyw)" \
 sn

dn: uid=user.13401,ou=People,dc=example,dc=com
sn: Zywiel

dn: uid=user.26820,ou=People,dc=example,dc=com
sn: Zywiel

dn: uid=user.40239,ou=People,dc=example,dc=com
sn: Zywiel

dn: uid=user.53658,ou=People,dc=example,dc=com
sn: Zywiel

dn: uid=user.67077,ou=People,dc=example,dc=com
sn: Zywiel

dn: uid=user.80496,ou=People,dc=example,dc=com
sn: Zywiel

dn: uid=user.93915,ou=People,dc=example,dc=com
sn: Zywiel
```

In this case, the server only requires an ordering index if it cannot reuse the (ordered) equality index instead. For example, if the equality index is encrypted, an ordering index must be maintained separately.

Virtual List View (Browsing) Index

A virtual list view (VLV) or browsing index is designed to help applications that list results. For example, a GUI application might let users browse through a list of users. VLVs help the server respond to clients that request server-side sorting of the search results.

VLV indexes correspond to particular searches. Configure your VLV indexes using the command line.

Extensible Matching Rule Index

In some cases, you need an index for a matching rule other than those described above.

For example, a generalized time-based matching index lets applications find entries with a time-based attribute later or earlier than a specified time.

Configure Indexes

You modify index configurations by using the **`dsconfig`** command. Configuration changes take effect after you rebuild the index with the new configuration, using the **`rebuild-index`** command. The **`dsconfig --help-database`** command lists subcommands for creating, reading, updating, and deleting index configuration.

TIP

Indexes are per directory backend rather than per base DN. To maintain separate indexes for different base DNs on the same server, put the entries in different backends.

Standard Indexes

New Index

The following example creates a new equality index for the `description` attribute:

```
$ dsconfig \
 create-backend-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name description \
 --set index-type:equality \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

```
  --trustStorePassword:file /path/to/opendj/config/keystore.pin \
  --no-prompt
```

Approximate Index

The following example adds an approximate index for the `sn` (surname) attribute:

```
$ dsconfig \
 set-backend-index-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name sn \
 --add index-type:approximate \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Approximate indexes depend on the Double Metaphone matching rule.

Extensible Match Index

DS servers support matching rules defined in LDAP RFCs. They also define DS-specific extensible matching rules.

The following are DS-specific extensible matching rules:

Name: `ds-mr-double-metaphone-approx`
: Double Metaphone Approximate Match described at http://aspell.net/metaphone/. The DS implementation always produces a single value rather than one or possibly two values.

  Configure approximate indexes as described in Approximate Index.

  For an example using this matching rule, see Approximate Match.

Name: `ds-mr-user-password-exact`
: User password exact matching rule used to compare encoded bytes of two hashed password values for exact equality.

Name: `ds-mr-user-password-equality`
: User password matching rule implemented as the user password exact matching rule.

Name: `partialDateAndTimeMatchingRule`

Partial date and time matching rule for matching parts of dates in time-based searches.

For an example using this matching rule, see Active Accounts.

Name: `relativeTimeOrderingMatch.gt`

Greater-than relative time matching rule for time-based searches.

For an example using this matching rule, see Active Accounts.

Name: `relativeTimeOrderingMatch.lt`

Less-than relative time matching rule for time-based searches.

For an example using this matching rule, see Active Accounts.

The following example configures an extensible matching rule index for "later than" and "earlier than" generalized time matching on a `lastLoginTime` attribute:

```
$ dsconfig \
 create-backend-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --set index-type:extensible \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.5 \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.6 \
 --index-name lastLoginTime \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Indexes for JSON

DS servers support attribute values that have JSON syntax. The following schema excerpt defines a `json` attribute with case-insensitive matching:

```
attributeTypes: ( json-attribute-oid NAME 'json'
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1 EQUALITY
caseIgnoreJsonQueryMatch
  SINGLE-VALUE X-ORIGIN 'DS Documentation Examples' )
```

When you index a JSON attribute defined in this way, the default directory server behavior is to maintain index keys for each JSON field. Large or numerous JSON objects can result in large indexes, which is wasteful. If you know which fields are used in search filters, you can choose to index only those fields.

As described in Schema and JSON, for some JSON objects only a certain field or fields matter when comparing for equality. In these special cases, the server can ignore other fields when checking equality during updates, and you would not maintain indexes for other fields.

How you index a JSON attribute depends on the matching rule in the attribute's schema definition, and on the JSON fields you expect to be used as search keys for the attribute.

Index JSON Attributes

The examples that follow demonstrate these steps:

1. Using the schema definition and the information in the following table, configure a custom schema provider for the attribute's matching rule, if necessary.

Matching Rule in Schema Definition	Fields in Search Filter	Custom Schema Provider Required?

`caseExactJsonQueryMatch`
 	
`caseIgnoreJsonQueryMatch`
	Any JSON field	No

`caseExactJsonQueryMatch`
 	
`caseIgnoreJsonQueryMatch`
	Specific JSON field or fields	Yes, see JSON Query Matching Rule Index

`caseExactJsonIdMatch`
 	
`caseIgnoreJsonIdMatch`
	"_id" field only	No, see JSON Query Matching Rule Index

Custom JSON equality matching rule	Specific field(s)	Yes, see JSON Equality Matching Rule Index

   A custom schema provider applies to all attributes using this matching rule.

2. Add the schema definition for the JSON attribute.

3. Configure the index for the JSON attribute.

4. Add the JSON attribute values in the directory data.

JSON Query Matching Rule Index

This example illustrates the steps in Index JSON Attributes.

NOTE

If you installed a directory server with the `ds-evaluation` profile, the custom index configuration is already present.

The following command configures a custom, case-insensitive JSON query matching rule. This only maintains keys for the `access_token` and `refresh_token` fields:

```
$ dsconfig \
 create-schema-provider \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --provider-name "Custom JSON Query Matching Rule" \
 --type json-query-equality-matching-rule \
 --set enabled:true \
 --set case-sensitive-strings:false \
 --set ignore-white-space:true \
 --set matching-rule-name:caseIgnoreJsonQueryMatch \
 --set matching-rule-oid:1.3.6.1.4.1.36733.2.1.4.1 \
 --set indexed-field:access_token \
 --set indexed-field:refresh_token \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

The following commands add schemas for a `json` attribute that uses the matching rule:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
```

```
 --bindPassword password << EOF
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( json-attribute-oid NAME 'json'
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1 EQUALITY
caseIgnoreJsonQueryMatch
  SINGLE-VALUE X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( json-object-class-oid NAME 'jsonObject' SUP top
  AUXILIARY MAY ( json ) X-ORIGIN 'DS Documentation Examples' )
EOF
```

The following command configures an index using the custom matching rule implementation:

```
$ dsconfig \
 create-backend-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name json \
 --set index-type:equality \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

For an example of how a client application could use this index, see JSON Query Filters.

JSON Equality Matching Rule Index

This example illustrates the steps in Index JSON Attributes.

NOTE

If you installed a directory server with the `ds-evaluation` profile, the custom index configuration is already present.

The following command configures a custom, case-insensitive JSON equality matching rule, `caseIgnoreJsonTokenIdMatch` . This indexes the `id` rather than the `_id` field:

```
$ dsconfig \
 create-schema-provider \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --provider-name "Custom JSON Token ID Matching Rule" \
 --type json-equality-matching-rule \
 --set enabled:true \
 --set case-sensitive-strings:false \
 --set ignore-white-space:true \
 --set matching-rule-name:caseIgnoreJsonTokenIDMatch \
 --set matching-rule-oid:1.3.6.1.4.1.36733.2.1.4.4.1 \
 --set json-keys:id \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Notice that this example defines a matching rule with OID `1.3.6.1.4.1.36733.2.1.4.4.1`. In production deployments, use a numeric OID allocated for your own organization.

The following commands add schemas for a `jsonToken` attribute, where the unique identifier is in the "id" field of the JSON object:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( jsonToken-attribute-oid NAME 'jsonToken'
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1 EQUALITY
caseIgnoreJsonTokenIDMatch
  SINGLE-VALUE X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( json-token-object-class-oid NAME
'JsonTokenObject' SUP top
```

```
  AUXILIARY MAY ( jsonToken ) X-ORIGIN 'DS Documentation Examples'
)
EOF
```

The following command configures an index using the custom matching rule implementation:

```
$ dsconfig \
 create-backend-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name jsonToken \
 --set index-type:equality \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

For an example of how a client application could use this index, see JSON Assertions.

Virtual List View Index

The following example shows how to create a VLV index. This example applies where GUI users browse user accounts, sorting on surname then given name:

```
$ dsconfig \
 create-backend-vlv-index \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name people-by-last-name \
 --set base-dn:ou=People,dc=example,dc=com \
 --set filter:"(|(givenName=*)(sn=*))" \
 --set scope:single-level \
 --set sort-order:"+sn +givenName" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

NOTE

NOTE

When referring to a VLV index after creation, you must add `vlv.` as a prefix. In other words, if you named the VLV index `people-by-last-name`, refer to it as `vlv.people-by-last-name` when rebuilding indexes, changing index properties such as the index entry limit, or verifying indexes.

VLV for Paged Server-Side Sort

A special VLV index lets the server sort results for a search that is technically *unindexed*. For example, users page through an entire directory database in a GUI. The user does not filter the data before seeing what is available.

Because the search is technically unindexed, the user performing the search must have the `unindexed-search` privilege.

The VLV index must have the following characteristics:

- Its filter must be "always true," `(&)`.
- Its scope must cover the search scope of the requests.
- Its base DN must match or be a parent of the base DN of the search requests.
- Its sort order must match the sort keys of the requests in the order they occur in the requests, starting with the first sort key used in the request. The VLV index sort order can include additional keys not present in a request.

  In other words, if the sort order of the VLV index is `+l +sn +cn`, then it works with requests having the following sort orders:

  - `+l +sn +cn`
  - `+l +sn`
  - `+l`

The following example commands demonstrate creating and using a VLV index to sort paged results by locality, surname, and then full name. The `l` attribute is not indexed by default. This example makes use of the **`rebuild-index`** command described below. The directory superuser is not subject to resource limits on the LDAP search operation:

```
$ dsconfig \
 create-backend-vlv-index \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name by-name \
 --set base-dn:ou=People,dc=example,dc=com \
```

```
 --set filter:"(&)" \
 --set scope:subordinate-subtree \
 --set sort-order:"+l +sn +cn" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ rebuild-index \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --baseDn dc=example,dc=com \
 --index vlv.by-name \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin

$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword password \
 --baseDn dc=example,dc=com \
 --sortOrder +l,+sn,+cn \
 --simplePageSize 5 \
 "(&)" \
 cn l sn

dn: uid=user.93953,ou=People,dc=example,dc=com
cn: Access Abedi
l: Abilene
sn: Abedi

dn: uid=user.40283,ou=People,dc=example,dc=com
cn: Achal Abernathy
l: Abilene
sn: Abernathy

dn: uid=user.67240,ou=People,dc=example,dc=com
cn: Alaine Alburger
l: Abilene
sn: Alburger
```

```
dn: uid=user.26994,ou=People,dc=example,dc=com
cn: Alastair Alexson
l: Abilene
sn: Alexson

dn: uid=user.53853,ou=People,dc=example,dc=com
cn: Alev Allen
l: Abilene
sn: Allen

Press RETURN to continue ^C
```

Rebuild Indexes

When you first import directory data, the directory server builds the indexes as part of the import process. DS servers maintain indexes automatically, updating them as directory data changes.

Only rebuild an index manually when it is necessary to do so. Rebuilding valid indexes wastes server resources, and is disruptive for client applications.

IMPORTANT

When you rebuild an index while the server is online, the index appears as degraded and unavailable while the server rebuilds it.

A search request that relies on an index in this state *may temporarily fail as an unindexed search*.

However, you must manually intervene when you:

- Create a new index for a new directory attribute.
- Create a new index for existing directory attribute.
- Change the server configuration in a way that affects the index, for example, by changing Index Entry Limits.
- Verify an existing index, and find that it has errors or is not in a valid state.

Automate Index Rebuilds

To automate the process of rebuilding indexes, use the `--rebuildDegraded` option. This rebuilds only degraded indexes, and does not affect valid indexes:

```
$ rebuild-index \
 --hostname localhost \
```

```
--port 4444 \
--bindDN uid=admin \
--bindPassword password \
--baseDN dc=example,dc=com \
--rebuildDegraded \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin
```

Clear a New Index for a New Attribute

When you add a new attribute, as described in Update LDAP Schema, and create an index for the new attribute, the new index appears as *degraded* or *invalid*. The attribute has not yet been used, and so the index is sure to be empty, not degraded.

In this case, you can safely use the **`rebuild-index --clearDegradedState`** command. The server can complete this operation quickly, because there is no need to scan the entire directory backend to rebuild a new, unused index.

In this example, an index has just been created for `newUnusedAttribute`. *If the newly indexed attribute has already been used, rebuild the index instead of clearing the degraded state.*

Before using the **`rebuild-index`** command, test the index status to make sure is has not been used:

1. DS servers must be stopped before you use the backendstat command:

   ```
   $ stop-ds
   ```

2. Check the status of the index(es).

   The third column of the output is the `Valid` column, which is `false` before the rebuild:

   ```
   $ backendstat \
    show-index-status \
    --backendID dsEvaluation \
    --baseDN dc=example,dc=com \
    | grep newUnusedAttribute

   newUnusedAttribute.presence                    ... false ...
   newUnusedAttribute.caseIgnoreMatch             ... false ...
   newUnusedAttribute.caseIgnoreSubstringsMatch:6 ... false ...
   ```

3. Update the index information to fix the value of the unused index:

```
$ rebuild-index \
 --offline \
 --baseDN dc=example,dc=com \
 --clearDegradedState \
 --index newUnusedAttribute
```

Alternatively, you can first start the server, and perform this operation while the server is online.

4. (Optional) With the server offline, check that the `Index Valid` column for the index status is now set to `true`:

```
$ backendstat \
 show-index-status \
 --backendID dsEvaluation \
 --baseDN dc=example,dc=com \
 | grep newUnusedAttribute

newUnusedAttribute.presence                    ... true ...
newUnusedAttribute.caseIgnoreMatch             ... true ...
newUnusedAttribute.caseIgnoreSubstringsMatch:6 ... true ...
```

5. Start the server if you have not done so already:

```
$ start-ds
```

Rebuild an Index

When you make a change that affects an index configuration, manually rebuild the index.

Individual indexes appear as degraded and are unavailable while the server rebuilds them. A search request that relies on an index in this state *may temporarily fail as an unindexed search*.

The following example rebuilds a degraded `cn` index immediately with the server online.

While the server is rebuilding the `cn` index, *search requests that would normally succeed may fail*:

```
$ rebuild-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
```

```
--bindPassword password \
--baseDN dc=example,dc=com \
--index cn \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin
```

Avoid Rebuilding All Indexes at Once

Rebuilding multiple non-degraded indexes at once is disruptive, and not recommended unless some change has affected all indexes.

If you use the `--rebuildAll` option, first take the backend offline, stop the server, or, at minimum, make sure that no applications connect to the server while it is rebuilding indexes.

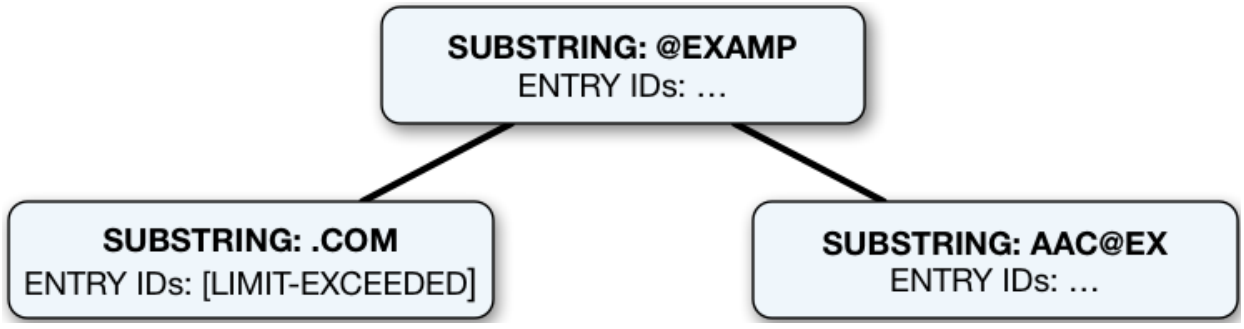
Index Entry Limits

An index is a tree of key-value pairs. The key is what the search is trying to match. The value is a list of entry IDs.

As the number of entries in the directory grows, the list of entry IDs for some keys can become very large. For example, every entry in the directory has `objectClass: top`. If the directory maintains a substring index for `mail`, the number of entries ending in `.com` could be huge.

A directory server therefore defines an *index entry limit*. When the number of entry IDs for a key exceeds the limit, the server stops maintaining a list of IDs for that key. The limit effectively makes a search using that key unindexed. Searches using other keys in the same index are not affected.

The following figure shows a fragment from a substring index for the `mail` attribute. The number of email addresses ending in `.com` has exceeded the index entry limit. For the other substring keys, the entry ID lists are still maintained. To save space, the entry IDs are not shown in the figure.



Ideally, the limit is set at the point where it becomes more expensive to maintain the entry ID list for a key, and to perform an indexed search than to perform an unindexed search. In practice, the limit is a trade off, with a default index entry limit value of 4000.

Check Index Entry Limits

The following steps show how to get information about indexes where the index entry limit is exceeded for some keys. In this case, the directory server holds 100,000 user entries. The settings for this directory server are reasonable.

Use the backendstat show-index-status command:

1. Stop DS servers before you use the **`backendstat`** command:

```
$ stop-ds
```

2. Non-zero values in the Over Entry Limit column of the output table indicate the number of keys for which the limit has been reached. The keys that are over the limit are then listed below the table:

```
$ backendstat show-index-status --backendID dsEvaluation -
-baseDN dc=example,dc=com

Index Name                                          ... Valid
... Over Entry Limit ...
...
mail.caseIgnoreIA5SubstringsMatch:6                 ... true
... 34 ...
...
givenName.caseIgnoreSubstringsMatch:6               ... true
... 9  ...
telephoneNumber.telephoneNumberSubstringsMatch:6 ... true
... 10 ...
...
sn.caseIgnoreSubstringsMatch:6                      ... true
... 14 ...
...
cn.caseIgnoreSubstringsMatch:6                      ... true
... 14 ...
objectClass.objectIdentifierMatch                   ... true
... 4  ...
...

Index:
/dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6
Over index-entry-limit keys: [.com] [0@mail] ...

Index:
```

```
/dc=com,dc=example/givenName.caseIgnoreSubstringsMatch:6
Over index-entry-limit keys: [a] [e] [i] [ie] [l] [n] [na]
[ne] [y]

Index:
/dc=com,dc=example/telephoneNumber.telephoneNumberSubstrin
gsMatch:6
Over index-entry-limit keys: [0] [1] [2] [3] [4] [5] [6]
[7] [8] [9]

Index: /dc=com,dc=example/cn.caseIgnoreSubstringsMatch:6
Over index-entry-limit keys: [a] [an] [e] [er] [i] [k] [l]
[n] [o] [on] [r] [s] [t] [y]

Index:
/dc=com,dc=example/objectClass.objectIdentifierMatch
Over index-entry-limit keys: [inetorgperson]
[organizationalperson] [person] [top]

Index: /dc=com,dc=example/sn.caseIgnoreSubstringsMatch:6
Over index-entry-limit keys: [a] [an] [e] [er] [i] [k] [l]
[n] [o] [on] [r] [s] [t] [y]
```

For example, every user entry has the object classes listed, and every user entry has an email address ending in `.com`, so those values are not specific enough to be used in search filters.

3. Start the server:

```
$ start-ds
```

Index Entry Limit Changes

In rare cases, the index entry limit might be too low for a certain key. This could manifest itself as a frequent, useful search becoming unindexed with no reasonable way to narrow the search.

You can change the index entry limit on a per-index basis. Do not do this in production unless you can explain and show why the benefits outweigh the costs.

IMPORTANT

Changing the index entry limit significantly can result in serious performance degradation. Be prepared to test performance thoroughly before you roll out an index entry limit change in production.

Consider a directory with more than 4000 groups in a backend. When the backend is brought online, a directory server searches for the groups with a search filter of `(|(objectClass=groupOfNames)(objectClass=groupOfEntries)(objectClass=groupOfUniqueNames))`, which is an unindexed search due to the default index entry limit setting. The following example raises the index entry limit for the `objectClass` index to `10000`. It rebuilds the index for the configuration change to take effect. The steps are the same for any other index:

```
$ dsconfig \
 set-backend-index-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name objectClass \
 --set index-entry-limit:10000 \
 --no-prompt \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin

$ stop-ds

$ rebuild-index \
 --baseDN dc=example,dc=com \
 --index objectClass \
 --offline

$ start-ds
```

It is also possible, but not recommended, to configure the global `index-entry-limit` for a backend. This changes the default for all indexes in the backend. Use the **`dsconfig set-backend-prop`** command:

```
# Not recommended
$ dsconfig \
 set-backend-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --set index-entry-limit:10000 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

```
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Verify Indexes

You can verify that indexes correspond to current directory data, and do not contain errors. Use the verify-index command.

The following example verifies the `cn` index offline:

```
$ stop-ds

$ verify-index \
 --baseDN dc=example,dc=com \
 --index cn \
 --clean \
 --countErrors
```

The output indicates whether any errors are found in the index.

Data Storage

DS directory servers store data in *backends* . A backend is a private server repository that can be implemented in memory, as an LDIF file, or as an embedded database.

Embedded database backends have these characteristics:

Suitable for large numbers of entries
: When creating a database backend, you choose the backend type. DS directory servers use JE backends for local data.

  The JE backend type is implemented using B-tree data structures. It stores data as key-value pairs, which is different from the model used by relational databases.

  JE backends are designed to hold hundreds of millions, or even billions of LDAP entries.

Fully managed by DS servers
: Let the server manage its backends and their database files.

  By default, backend database files are located under the `opendj/db` directory.

WARNING

WARNING

Do not compress, tamper with, or otherwise alter backend database files directly, unless specifically instructed to do so by a qualified ForgeRock technical support engineer. External changes to backend database files can render them unusable by the server.

If you use snapshot technology for backup, read Back Up Using Snapshots and Restore From a Snapshot.

DS servers provide the **`dsbackup`** command for backing up and restoring database backends. For details, see Backup and Restore.

Self-cleaning

A JE backend stores data on disk using append-only log files with names like `number.jdb`. The JE backend writes updates to the highest-numbered log file. The log files grow until they reach a specified size (default: 1 GB). When the current log file reaches the specified size, the JE backend creates a new log file.

To avoid an endless increase in database size on disk, JE backends clean their log files in the background. A cleaner thread copies active records to new log files. Log files that no longer contain active records are deleted.

Due to the cleanup processes, JE backends can actively write to disk, even when there are no pending client or replication operations.

Configurable I/O behavior

By default, JE backends let the operating system potentially cache data for a period of time before flushing the data to disk. This setting trades full durability with higher disk I/O for good performance with lower disk I/O.

With this setting, it is possible to lose the most recent updates that were not yet written to disk, in the event of an underlying OS or hardware failure.

You can modify this behavior by changing the advanced configuration settings for the JE backend. If necessary, you can change the advanced setting, db-durability, using the **`dsconfig set-backend-prop`** command.

Automatic recovery

When a JE backend is opened, it recovers by recreating its B-tree structure from its log files.

This is a normal process. It lets the backend recover after an orderly shutdown or after a crash.

Automatic verification

JE backends run checksum verification periodically on the database log files.

If the verification process detects backend database corruption, then the server logs an error message and takes the backend offline. If this happens, restore the corrupted backend from backup so that it can be used again.

By default, the verification runs every night at midnight local time. If necessary, you can change this behavior by adjusting the advanced settings, db-run-log-verifier and db-log-verifier-schedule, using the **`dsconfig set-backend-prop`** command.

Encryption support

JE backends support encryption for data confidentiality.

For details, see Data Encryption.

Depending on the setup profiles used at installation time, DS servers have backends with the following default settings:

Backend	Type	Optional? (1)	Replicated?	Part of Backup?	Details
adminRoot	LDIF	No	If enabled	If enabled	Symmetric keys for (deprecated) reversible password storage schemes. Base DN: `cn=admin data` Base directory: `db/adminRoot`
amCts	JE	Yes	Yes	Yes	AM core token store (CTS) data. More information: Install DS for AM CTS. Base DN: `ou=tokens` Base directory: `db/amCts`

Backend	Type	Optional? (1)	Replicated?	Part of Backup?	Details
amIdentityStore	JE	Yes	Yes	Yes	AM identities. More information: Install DS for AM Identities. Base DN: `ou=identities` Base directory: `db/amIdentityStore`
cfgStore	JE	Yes	Yes	Yes	AM configuration, policy, and other data. More information: Install DS for AM Configuration. Base DN: `ou=am-config` Base directory: `db/cfgStore`
changelogDb	Changelog	Yes(2)	N/A	No	Change data for replication and change notifications. More information: Changelog for Notifications. Base DN: `cn=changelog` Base directory: `changelogDb`

Backend	Type	Optional? (1)	Replicated?	Part of Backup?	Details
config	Config	No	N/A	No	File-based representation of this server's configuration. Do not edit `config/config.ldif` directly. Use the **`dsconfig`** command instead. Base DN: `cn=config` Base directory: `config`
dsEvaluation	JE	Yes	Yes	Yes	Example.com sample data. More information: Install DS for Evaluation. Base DN: `dc=example,dc=com` Base directory: `db/dsEvaluation`
idmRepo	JE	Yes	Yes	Yes	IDM repository data. More information: Install DS as an IDM Repository. Base DN: `dc=openidm,dc=forgerock,dc=com` Base directory: `db/idmRepo`

Backend	Type	Optional? (1)	Replicated?	Part of Backup?	Details
monitor	Monitor	No	N/A	N/A	Single entry; volatile monitoring metrics maintained in memory since server startup. More information: LDAP-Based Monitoring. Base DN: `cn=monitor` Base directory: N/A
monitorUser	LDIF	Yes	Yes	Yes	Single entry; default monitor user account. Base DN: `uid=Monitor` Base directory: `db/monitorUser`
rootUser	LDIF	No(3)	No	Yes	Single entry; default directory superuser account. Base DN: `uid=admin` Base directory: `db/rootUser`

Backend	Type	Optional? (1)	Replicated?	Part of Backup?	Details
root DSE	Root DSE	No	N/A	N/A	Single entry describing server capabilities. Use **`ldapsearch --baseDn "" --searchScope base "(&)" +`** to read all the (operational) attributes of this entry. Base DN: `""` (empty string) Base directory: N/A
schema	Schema	No	Yes	Yes	Single entry listing LDAP schema definitions. More information: LDAP Schema. Base DN: `cn=schema` Base directory: `db/schema`
tasks	Task	No	N/A	Yes	Scheduled tasks for this server. Use the **`manage-tasks`** command. Base DN: `cn=tasks` Base directory: `db/tasks`

Backend	Type	Optional? (1)	Replicated?	Part of Backup?	Details
userData	JE	Yes	Yes	Yes	User entries imported from the LDIF you provide. More information: Install DS for User Data. Base directory: `db/userData`

(1) Optional backends depend on the setup choices.

(2) The changelog backend is mandatory for servers with a replication server role.

(3) You must create a superuser account at setup time. You may choose to replace it later. For details, see Use a Non-Default Superuser Account.

Create a Backend

When you create a new backend on a replicated directory server, let the server replicate the new data:

1. Configure the new backend.

   The following example creates a database backend for Example.org data. The backend relies on a JE database for data storage and indexing:

```
$ dsconfig \
 create-backend \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --backend-name exampleOrgBackend \
 --type je \
 --set enabled:true \
 --set base-dn:dc=example,dc=org \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

When you create a new backend using the **`dsconfig`** command, DS directory servers create the following indexes automatically:

Index	Approx.	Equality	Ordering	Presence	Substring	Entry Limit
`aci`	-	-	-	Yes	-	4000
`dn2id`	Non-configurable internal index					
`ds-sync-conflict`	-	Yes	-	-	-	4000
`ds-sync-hist`	-	-	Yes	-	-	4000
`entryUUID`	-	Yes	-	-	-	4000
`id2children`	Non-configurable internal index					
`id2subtree`	Non-configurable internal index					
`objectClass`	-	Yes	-	-	-	4000

2. Verify that replication is enabled:

```
$ dsconfig \
 get-synchronization-provider-prop \
 --provider-name "Multimaster Synchronization" \
 --property enabled \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
Property : Value(s)
---------:---------
enabled  : true
```

If replication should be enabled but is not, use `dsconfig set-synchronization-provider-prop --set enabled:true` to enable it.

3. Let the server replicate the base DN of the new backend:

```
$ dsconfig \
 create-replication-domain \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --domain-name dc=example,dc=org \
 --type generic \
 --set enabled:true \
 --set base-dn:dc=example,dc=org \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

4. If you have existing data for the backend, follow the appropriate procedure to initialize replication.

   For details, see Manual Initialization.

If you must temporarily disable replication for the backend, take care to avoid losing changes. For details, see Disable Replication.

Import and Export

The following procedures demonstrate how to import and export LDIF data.

For details on creating custom sample LDIF to import, refer to Generate test data.

Import LDIF

If you are initializing replicas by importing LDIF, refer to Initialize from LDIF for context.

IMPORTANT

IMPORTANT

- Importing from LDIF overwrites all data in the target backend with entries from the LDIF data.
- If the LDIF was exported from another server, it may contain pre-encoded passwords. As long as DS supports the password storage schemes used to encode the passwords, you can enable the storage schemes in the configuration to migrate existing passwords into DS. For details, see Password Storage.

  By default, password policies do not allow you to use pre-encoded passwords. You can change this behavior by changing the default password policy configuration property, `allow-pre-encoded-passwords`.

  The DS import process does not warn you about passwords that use disabled password storage schemes. Instead, search the LDIF to find all the password storage schemes in use, and make sure all the schemes are enabled in the server configuration. Users whose passwords are stored with a disabled scheme cannot bind successfully.
- LDIF from another server can include passwords encrypted with a reversible storage scheme, such as AES or Blowfish. To decrypt the passwords, the server must use the same deployment key as the server that encrypted the passwords.

Perform either of the following steps to import `dc=example,dc=com` data into the `dsEvaluation` backend:

- To import offline, shut down the server before you run the **`import-ldif`** command:

```
$ stop-ds
$ import-ldif \
 --offline \
 --backendId dsEvaluation \
 --includeBranch dc=example,dc=com \
 --ldifFile example.ldif
```

- To import online, schedule a task:

```
$ start-ds
$ import-ldif \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --backendId dsEvaluation \
```

```
 --includeBranch dc=example,dc=com \
 --ldifFile example.ldif \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
```

You can schedule the import task to start at a particular time using the `--start` option.

Export LDIF

Perform either of the following steps to export `dc=example,dc=com` data from the `dsEvaluation` backend:

- To export offline, shut down the server before you run the **`export-ldif`** command:

```
$ stop-ds
$ export-ldif \
 --offline \
 --backendId dsEvaluation \
 --includeBranch dc=example,dc=com \
 --ldifFile backup.ldif
```

- To export online, schedule a task:

```
$ export-ldif \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --backendId dsEvaluation \
 --includeBranch dc=example,dc=com \
 --ldifFile backup.ldif \
 --start 0 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
```

The `--start 0` option tells the directory server to start the export task immediately.

You can specify a time for the task to start using the format yyyymmddHHMMSS. For example, `20250101043000` specifies a start time of 4:30 AM on January 1, 2025.

If the server is not running at the time specified, it attempts to perform the task after it restarts.

Split Data

Splitting data impacts replication, and can require that you interrupt the service.

On one set of replicas, create the `exampleData` backend to hold all `dc=example,dc=com` data except `ou=People,dc=example,dc=com`:

```
# Create a backend for the data not under the subordinate base DN:
$ dsconfig \
 create-backend \
 --backend-name exampleData \
 --type je \
 --set enabled:true \
 --set base-dn:dc=example,dc=com \
 --offline \
 --no-prompt

# Let the server replicate the base DN of the new backend:
$ dsconfig \
 create-replication-domain \
 --provider-name "Multimaster Synchronization" \
 --domain-name dc=example,dc=com \
 --type generic \
 --set enabled:true \
 --set base-dn:dc=example,dc=com \
 --offline \
 --no-prompt

# Import data not under the subordinate base DN:
$ import-ldif \
 --backendId exampleData \
 --excludeBranch ou=People,dc=example,dc=com \
 --ldifFile example.ldif \
 --offline
```

On another set of replicas, create a `peopleData` backend to hold `ou=People,dc=example,dc=com` data:

```
# Create a backend for the data under the subordinate base DN:
$ dsconfig \
 create-backend \
 --backend-name peopleData \
 --type je \
 --set enabled:true \
```

```
 --set base-dn:ou=People,dc=example,dc=com \
 --offline \
 --no-prompt

# Let the server replicate the base DN of the new backend:
$ dsconfig \
 create-replication-domain \
 --provider-name "Multimaster Synchronization" \
 --domain-name ou=People,dc=example,dc=com \
 --type generic \
 --set enabled:true \
 --set base-dn:ou=People,dc=example,dc=com \
 --offline \
 --no-prompt

# Import data under the subordinate base DN:
$ import-ldif \
 --backendId peopleData \
 --includeBranch ou=People,dc=example,dc=com \
 --ldifFile example.ldif \
 --offline
```

After completing the data import process, start the replicas.

To split an *existing* backend, follow these high-level steps:

1. Stop the affected replica.
2. Create the new backend.
3. Export the data for the backend.
4. Import the data under the subordinate base DN into the new backend, using the `--includeBranch` option.
5. Delete all data under the subordinate base DN from the old backend.
6. Start the affected replica.

Disk Space Thresholds

Directory data growth depends on applications that use the directory. When directory applications add more data than they delete, the database backend grows until it fills the available disk space. The system can end up in an unrecoverable state if no disk space is available.

Database backends therefore have advanced properties, `disk-low-threshold` and `disk-full-threshold`. When available disk space falls below `disk-low-threshold`, the directory server only allows updates from users and applications that have the

`bypass-lockdown` privilege. When available space falls below `disk-full-threshold`, the directory server stops allowing updates, instead returning an `UNWILLING_TO_PERFORM` error to each update request.

If growth across the directory service tends to happen quickly, set the thresholds higher than the defaults to allow more time to react when growth threatens to fill the disk. The following example sets `disk-low-threshold` to 10 GB `disk-full-threshold` to 5 GB for the `dsEvaluation` backend:

```
$ dsconfig \
 set-backend-prop \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --set "disk-low-threshold:10 GB" \
 --set "disk-full-threshold:5 GB" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

The properties disk-low-threshold, and disk-full-threshold are listed as *advanced* properties.

To examine their values with the **`dsconfig`** command, use the `--advanced` option:

```
$ dsconfig \
 get-backend-prop \
 --advanced \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --property disk-low-threshold \
 --property disk-full-threshold \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

Property            : Value(s)
--------------------:---------
disk-full-threshold : 5 gb
disk-low-threshold  : 10 gb
```

Entry Expiration

If the directory service creates many entries that expire and should be deleted, you could find the entries with a time-based search and then delete them individually. That approach uses replication to delete the entries in all replicas. It has the disadvantage of generating potentially large amounts of replication traffic.

Entry expiration lets you configure the backend database to delete the entries as they expire. This approach deletes expired entries at the backend database level, without generating replication traffic. AM uses this approach when relying on DS to delete expired token entries, as demonstrated in Install DS for AM CTS.

Backend indexes for generalized time (timestamp) attributes have these properties to configure automated, optimized entry expiration and removal:

- ttl-enabled
- ttl-age

Configure this capability by performing the following steps:

1. Prepare an ordering index for a generalized time (timestamp) attribute on entries that expire.

   For details, see Configure Indexes and Active Accounts.

2. Using the **`dsconfig set-backend-index-prop`** command, set `ttl-enabled` on the index to true, and set `ttl-age` on the index to the desired entry lifetime duration.

3. Optionally enable the access log to record messages when the server deletes an expired entry.

   Using the **`dsconfig set-log-publisher-prop`** command, set `suppress-internal-operations:false` for the access log publisher. Note that this causes the server to log messages for all internal operations.

   When the server deletes an expired entry, it logs a message with `"additionalItems":{"ttl": null}` in the response.

Once you configure and build the index, the backend can delete expired entries. At intervals of 10 seconds, the backend automatically deletes entries whose timestamps on the attribute are older than the specified lifetime. Entries that expire in the interval between deletions are removed on the next round. Client applications should therefore check that entries have not expired, as it is possible for expired entries to remain available until the next round of deletions.

When using this capability, keep the following points in mind:

- Entry expiration is per index. The time to live depends on the value of the indexed attribute and the `ttl-age` setting, and all matching entries are subject to TTL.
- If multiple indexes' `ttl-enabled` and `ttl-age` properties are configured, as soon as one of the entry's matching attributes exceeds the TTL, the entry is eligible for deletion.
- The backend deletes the entries directly as an internal operation. The server only records the deletion when `suppress-internal-operations: false` for the access log publisher. Persistent searches do not return the deletion.

  Furthermore, this means that *deletion is not replicated*. To ensure expired entries are deleted on all replicas, use the same indexes with the same settings on all replicas.
- When a backend deletes an expired entry, the effect is a subtree delete. In other words, if a parent entry expires, the parent entry *and all the parent's child entries* are deleted.

  If you do not want parent entries to expire, index a generalized time attribute that is only present on its child entries.
- The backend deletes expired entries atomically.

  If you update the TTL attribute to prevent deletion and the update succeeds, then TTL has effectively been reset.
- Expiration fails when the `index-entry-limit` is exceeded. (For background information, see Index Entry Limits.)

  This only happens if the timestamp for the indexed attribute matches to the nearest millisecond on more than 4000 entries (for default settings). This corresponds to four million timestamp updates per second, which would be very difficult to reproduce in a real directory service.

  It is possible, however, to construct and import an LDIF file where more than 4000 entries have the same timestamp. Make sure not to reuse the same timestamp for thousands of entries when artificially creating entries that you intend to expire.

Add a Base DN to a Backend

The following example adds the base DN `o=example` to the `dsEvaluation` backend, and creates a replication domain configuration to replicate the data under the new base DN:

```
$ dsconfig \
 set-backend-prop \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
```

```
 --bindPassword password \
 --backend-name dsEvaluation \
 --add base-dn:o=example \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 get-backend-prop \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --property base-dn \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
Property : Value(s)
---------:---------------------------------
base-dn  : "dc=example,dc=com", o=example

$ dsconfig \
 create-replication-domain \
 --provider-name "Multimaster Synchronization" \
 --domain-name o=example \
 --type generic \
 --set enabled:true \
 --set base-dn:o=example \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

When you add a base DN to a backend, the base DN must not be subordinate to any other base DNs in the backend. As in the commands shown here, you can add the base DN `o=example` to a backend that already has a base DN `dc=example,dc=com`. You cannot, however, add `o=example,dc=example,dc=com` as a base DN, because that is a child of `dc=example,dc=com`.

Delete a Backend

When you delete a database backend with the **`dsconfig delete-backend`** command, the directory server does not actually remove the database files for these reasons:

- A mistake could potentially cause lots of data to be lost.
- Deleting a large database backend could cause severe service degradation due to a sudden increase in I/O load.

After you run the **`dsconfig delete-backend`** command, manually remove the database backend files, and remove replication domain configurations any base DNs you deleted by removing the backend.

If run the **`dsconfig delete-backend`** command by mistake, but have not yet deleted the actual files, recover the data by creating the backend again, and reconfiguring and rebuilding the indexes.

Groups

DS servers support several methods of grouping entries:

- Dynamic Groups look up their membership based on an LDAP filter.
- Static Groups list each member.

  Static groups are easy to start, but can become large and expensive to maintain.

  For static groups, you must have a mechanism to remove members whose entries are deleted, and members whose entries are modified in a way that ends their membership. DS servers use a *referential integrity* plugin for this.
- Virtual Static Groups use a dynamic group-style definition, but let applications list group members as if the group were static.

TIP

Tip

The examples that follow assume `ou=Groups,dc=example,dc=com` already exists. The `ds-evaluation` profile includes this entry by default. If you are using another profile, you can create the groups entry:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: ou=Groups,dc=example,dc=com
objectClass: organizationalunit
objectClass: top
ou: Groups
EOF
```

Dynamic Groups

A *dynamic group* references members using LDAP URLs. Dynamic group entries take the `groupOfURLs` object class, with one or more `memberURL` values specifying LDAP URLs to identify group members.

Dynamic groups are a natural fit for directory servers. If you have a choice, choose dynamic groups over static groups for these reasons:

- Dynamic group membership is a property of a member's entry.

  There is no need to maintain a separate entry with the list of members.
- Determining dynamic group membership is as simple as applying the member URL criteria.
- Dynamic groups scale to any size without performance issues.

  Dynamic group entries remain small even when the group has a large number of members.

The following dynamic group includes entries matching the filter `"(l=San Francisco)"` (users whose location is San Francisco):

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

```
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com
cn: My Dynamic Group
objectClass: top
objectClass: groupOfURLs
ou: Groups
memberURL: ldap:///ou=People,dc=example,dc=com??sub?l=San
Francisco
EOF
```

Group membership changes dynamically as entries change to match the `memberURL` values:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(&(uid=*jensen)(isMemberOf=cn=My Dynamic
Group,ou=Groups,dc=example,dc=com))" \
 1.1

dn: uid=bjensen,ou=People,dc=example,dc=com

dn: uid=rjensen,ou=People,dc=example,dc=com

$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: uid=ajensen,ou=People,dc=example,dc=com
changetype: modify
replace: l
l: San Francisco
EOF

$ ldapsearch \
```

```
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(&(uid=*jensen)(isMemberOf=cn=My Dynamic
Group,ou=Groups,dc=example,dc=com))" \
 1.1

dn: uid=ajensen,ou=People,dc=example,dc=com

dn: uid=bjensen,ou=People,dc=example,dc=com

dn: uid=rjensen,ou=People,dc=example,dc=com
```

Virtual Static Groups

DS servers let you create *virtual static groups*. Virtual static groups allow applications to see dynamic groups as if they had an enumerated list of members like a static group.

The virtual static group takes the auxiliary object class `ds-virtual-static-group`. Virtual static groups use the object class `groupOfNames`, or `groupOfUniqueNames`. Instead of `member` or `uniqueMember` attributes, they have `ds-target-group-dn` attributes pointing to other groups.

Generating the list of members can be resource-intensive for large groups. By default, you cannot retrieve the list of members. If you have an application that must read the list of members, change the configuration of `Virtual Static member` or `Virtual Static uniqueMember` to set `allow-retrieving-membership:true`:

```
$ dsconfig \
 set-virtual-attribute-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --name "Virtual Static member" \
 --set allow-retrieving-membership:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

The following example creates a virtual static group, and reads the group entry with all members:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: cn=Virtual Static,ou=Groups,dc=example,dc=com
cn: Virtual Static
objectclass: top
objectclass: groupOfNames
objectclass: ds-virtual-static-group
ds-target-group-dn: cn=My Dynamic
Group,ou=Groups,dc=example,dc=com
EOF

$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(cn=Virtual Static)"

dn: cn=Virtual Static,ou=Groups,dc=example,dc=com
objectClass: top
objectClass: groupOfNames
objectClass: ds-virtual-static-group
cn: Virtual Static
ds-target-group-dn: cn=My Dynamic
Group,ou=Groups,dc=example,dc=com
member: uid=abergin,ou=People,dc=example,dc=com
member: uid=ajensen,ou=People,dc=example,dc=com
member: uid=aknutson,ou=People,dc=example,dc=com
member: uid=awalker,ou=People,dc=example,dc=com
member: uid=aworrell,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=bplante,ou=People,dc=example,dc=com
member: uid=btalbot,ou=People,dc=example,dc=com
member: uid=cwallace,ou=People,dc=example,dc=com
```

```
member: uid=dakers,ou=People,dc=example,dc=com
member: uid=dthorud,ou=People,dc=example,dc=com
member: uid=ewalker,ou=People,dc=example,dc=com
member: uid=gfarmer,ou=People,dc=example,dc=com
member: uid=jbourke,ou=People,dc=example,dc=com
member: uid=jcampaig,ou=People,dc=example,dc=com
member: uid=jmuffly,ou=People,dc=example,dc=com
member: uid=jreuter,ou=People,dc=example,dc=com
member: uid=jwalker,ou=People,dc=example,dc=com
member: uid=kcarter,ou=People,dc=example,dc=com
member: uid=kschmith,ou=People,dc=example,dc=com
member: uid=mjablons,ou=People,dc=example,dc=com
member: uid=mlangdon,ou=People,dc=example,dc=com
member: uid=mschneid,ou=People,dc=example,dc=com
member: uid=mtalbot,ou=People,dc=example,dc=com
member: uid=mtyler,ou=People,dc=example,dc=com
member: uid=mwhite,ou=People,dc=example,dc=com
member: uid=pshelton,ou=People,dc=example,dc=com
member: uid=rjensen,ou=People,dc=example,dc=com
member: uid=smason,ou=People,dc=example,dc=com
member: uid=tlabonte,ou=People,dc=example,dc=com
member: uid=tschmith,ou=People,dc=example,dc=com
```

Static Groups

A *static group* entry enumerates the entries in the group. Static group entries grow as their membership increases.

Large static groups are a performance bottleneck. If you have a choice, choose dynamic groups over static groups for these reasons:

- Static group membership is defined in a separate, potentially large LDAP entry.

  Large static group entries are expensive to maintain, cache, and replicate.
- Determining static group membership involves reading the group entry, or using virtual membership attributes.
- Static group performance tends to decrease with size.

  Reading and updating the group entry becomes more expensive as group size grows.

IMPORTANT

IMPORTANT

When working with static groups, also read Verify Membership and Referential Integrity.

If you have a deployment with large static groups that are updated regularly, use an entry cache. For details, see Cache for Large Groups.

Static group entries are based on one of the following standard object classes:

groupOfNames
: The `groupOfNames` object class requires at least one `member` attribute.

  Each value is the distinguished name of an entry.

groupOfEntries
: The `groupOfEntries` object class requires zero or more `member` attributes.

groupOfUniqueNames
: The `groupOfUniqueNames` object class has at least one `uniqueMember` attribute.

  Each value follows Name and Optional UID syntax. Name and Optional UID syntax values are a DN, optionally followed by `\#BitString`. The BitString, such as `'0101111101'B`, serves to distinguish the entry from another entry with the same DN, which can occur when the original entry was deleted and a new entry was created with the same DN.

Like other LDAP attributes, each group member attribute value is unique. LDAP does not allow duplicate values for the same attribute on the same entry.

TIP

When creating a group entry, use `groupOfNames` or `groupOfEntries` where possible.

To create a static group, add a group entry such as the following to the directory:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
cn: My Static Group
objectClass: groupOfNames
objectClass: top
```

```
ou: Groups
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
EOF
```

To change group membership, modify the values of the membership attribute:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
changetype: modify
add: member
member: uid=scarter,ou=People,dc=example,dc=com
EOF

$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(cn=My Static Group)"

dn: cn=My Static Group,ou=Groups,dc=example,dc=com
objectClass: groupOfNames
objectClass: top
cn: My Static Group
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
member: uid=scarter,ou=People,dc=example,dc=com
ou: Groups
```

RFC 4519 says a `groupOfNames` entry must have at least one member. Although DS servers allow you to create a `groupOfNames` without members, strictly speaking, that

behavior is not standard. Alternatively, you can use the `groupOfEntries` object class as shown in the following example:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
cn: Initially Empty Static Group
objectClass: groupOfEntries
objectClass: top
ou: Groups
EOF

$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
changetype: modify
add: member
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
member: uid=scarter,ou=People,dc=example,dc=com
EOF
```

Nested Groups

DS servers let you nest groups. The following example shows a group of groups of managers and administrators:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
```

```
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: cn=The Big Shots,ou=Groups,dc=example,dc=com
cn: The Big Shots
objectClass: groupOfNames
objectClass: top
ou: Groups
member: cn=Accounting Managers,ou=groups,dc=example,dc=com
member: cn=Directory Administrators,ou=Groups,dc=example,dc=com
member: cn=HR Managers,ou=groups,dc=example,dc=com
member: cn=PD Managers,ou=groups,dc=example,dc=com
member: cn=QA Managers,ou=groups,dc=example,dc=com
EOF
```

Although not shown in the example above, DS servers let you nest groups within nested groups.

DS servers let you create dynamic groups of groups. The following example shows a group of other groups. The members of this group are themselves groups, not users:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: cn=Group of Groups,ou=Groups,dc=example,dc=com
cn: Group of Groups
objectClass: top
objectClass: groupOfURLs
ou: Groups
memberURL: ldap:///ou=Groups,dc=example,dc=com??sub?ou=Groups
EOF
```

Use the `isMemberOf` attribute to determine what groups a member belongs to, as described in Verify Membership. The following example requests the groups that Kirsten Vaughan belongs to:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
```

```
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=kvaughan)" \
 isMemberOf

dn: uid=kvaughan,ou=People,dc=example,dc=com
isMemberOf: cn=Directory
Administrators,ou=Groups,dc=example,dc=com
isMemberOf: cn=HR Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
```

Notice that Kirsten is a member of The Big Shots group.

Notice also that Kirsten does not belong to the Group of Groups. The members of that group are groups, not users. The following example requests the groups that the directory administrators group belongs to:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(cn=Directory Administrators)" \
 isMemberOf

dn: cn=Directory Administrators,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
```

The following example shows which groups each group belong to. The search is unindexed, and so is performed here with directory administrator credentials:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN dc=example,dc=com \
```

```
 "(ou=Groups)" \
 isMemberOf

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

dn: cn=Accounting Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=Directory Administrators,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=HR Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com

dn: cn=My Static Group,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=PD Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=QA Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=The Big Shots,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
```

Notice that the group of groups is not a member of itself.

Verify Membership

For a dynamic group, you can check membership directly on the candidate member's entry using the same search criteria as the dynamic group's member URL.

DS servers also let you verify which groups a user belongs to by requesting the virtual `isMemberOf` attribute. This is more efficient than reading a large static group to determine whether a candidate belongs to the group:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 isMemberOf

dn: uid=bjensen,ou=People,dc=example,dc=com
isMemberOf: cn=Initially Empty Static
Group,ou=Groups,dc=example,dc=com
isMemberOf: cn=My Static Group,ou=Groups,dc=example,dc=com
isMemberOf: cn=My Dynamic Group,ou=Groups,dc=example,dc=com
isMemberOf: cn=Virtual Static,ou=Groups,dc=example,dc=com
isMemberOf: cn=Carpoolers,ou=Self
Service,ou=Groups,dc=example,dc=com
```

You must request `isMemberOf` explicitly.

Referential Integrity

When you delete or rename an entry that belongs to static groups, that entry's DN must be removed or changed in each group it belongs to. You can configure the server to resolve membership changes by enabling referential integrity.

Referential integrity functionality is implemented as a plugin. The referential integrity plugin is disabled by default. To enable the plugin, use the **`dsconfig`** command:

```
$ dsconfig \
 set-plugin-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --plugin-name "Referential Integrity" \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

```
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

With the plugin enabled, referential integrity resolves group membership automatically:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(cn=My Static Group)"

dn: cn=My Static Group,ou=Groups,dc=example,dc=com
objectClass: groupOfNames
objectClass: top
cn: My Static Group
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
member: uid=scarter,ou=People,dc=example,dc=com
ou: Groups

$ ldapdelete \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 uid=scarter,ou=People,dc=example,dc=com

$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(cn=My Static Group)"

dn: cn=My Static Group,ou=Groups,dc=example,dc=com
objectClass: groupOfNames
```

```
objectClass: top
cn: My Static Group
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
ou: Groups
```

By default, the referential integrity plugin is configured to manage `member` and `uniqueMember` attributes. These attributes take values that are DNs, and are indexed for equality by default for the default backend. Before you add an additional attribute to manage, make sure that it has DN syntax and that it is indexed for equality. DS servers require indexes because an unindexed search can consume too many of the server's resources. For instructions on indexing attributes, see Configure Indexes.

Consider these settings when configuring the referential integrity plugin:

- `check-references:true` checks that members' entries exist when added to a group.
- `check-references-filter-criteria` lets your members' entries match an LDAP filter.

  For example, `check-references-filter-criteria:member:(objectclass=person)` checks that members are person entries.
- `check-references-scope-criteria:naming-context` checks that members' entries are in the same naming context (base DN).

Virtual Attributes

Virtual attributes augment directory entries with attribute values that the DS server computes or obtains dynamically. Virtual attribute values do not exist in persistent storage. They help to limit the amount of data that needs to be stored. They fit well in some use cases, such as determining the groups a users belongs to, or adding an ETag to an entry.

NOTE

NOTE

- Do not index virtual attributes.

  Virtual attribute values are generated by the server when they are read. They are not designed to be stored in a persistent index. Since you do not index virtual attributes, searching on a virtual attribute can result in an unindexed search. Unindexed searches are resource-intensive for large directories. By default, a directory server lets only the directory superuser perform unindexed searches.

- Avoid searches that use a simple filter with a virtual attribute.

  If you must use a virtual attribute in a search filter, use it in a complex search filter that first narrows the search by filtering on an indexed attribute. For example, the following filter first narrows the search based on the user's ID before checking group membership. Make sure that the user performing the search has access to read `isMemberOf` in the results:

  ```
  (&(uid=user-id)(isMemberOf=group-dn))
  ```

If you must use the `entryDN` and `isMemberOf` virtual attributes in a filter, use a simple equality filter. The following example shows how to add access to read `isMemberOf`, and then run a search that returns the common names for members of a group:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr="isMemberOf")(version 3.0; acl "See isMemberOf";
allow (read,search,compare)
  groupdn= "ldap:///cn=Directory
Administrators,ou=Groups,dc=example,dc=com";)
EOF

$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
```

```
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(isMemberOf=cn=Directory
Administrators,ou=Groups,dc=example,dc=com)" \
 cn

dn: uid=hmiller,ou=People,dc=example,dc=com
cn: Harry Miller

dn: uid=kvaughan,ou=People,dc=example,dc=com
cn: Kirsten Vaughan

dn: uid=rdaugherty,ou=People,dc=example,dc=com
cn: Robert Daugherty
```

DS servers define the following virtual attributes by default:

entryDN
: The DN of the entry.

entryUUID
: String. Provides a universally unique identifier for the entry.

etag
: String entity tag. Defined in RFC 2616. Useful when checking whether an entry has changed since it was retrieved.

hasSubordinates
: Boolean. Indicates whether the entry has children.

numSubordinates
: The number of child entries directly beneath the entry.

isMemberOf
: DN. Groups the entry belongs to.

  By default, the server generates `isMemberOf` on entries having one of the following object classes:

  - `groupOfEntries`
  - `groupOfNames`
  - `groupOfUniqueNames`
  - `person`

To generate `isMemberOf` on entries with other object classes, edit the filter property of the `isMemberOf` virtual attribute configuration.

member
DN. Generated for virtual static groups.

uniqueMember
DN. Generated for virtual static groups.

pwdPolicySubentry
DN of the password policy that applies to the entry.

Default global access control settings prevent normal users from accessing this operational attribute.

subschemaSubentry
DN. References the schema definitions.

collectiveAttributeSubentries
DNs of applicable collective attribute definitions.

governingStructureRule
DN. References the rule specifying the type of subordinates the entry can have.

structuralObjectClass
DN. References the structural object class for the entry.

Virtual attributes are generally operational, and so are returned only when explicitly requested. The searches in these examples are unindexed, and so are performed with directory administrator credentials:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(dc=example)"

dn: dc=example,dc=com
dc: example
objectClass: domain
objectClass: top

$ ldapsearch \
```

```
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(dc=example)" \
 numSubordinates

dn: dc=example,dc=com
numSubordinates: 12
```

You can use the existing virtual attribute types to create your own virtual attributes. You can also use the `user-defined` type to create your own virtual attribute types. The virtual attribute is defined by the server configuration, which is not replicated:

```
$ dsconfig \
 create-virtual-attribute \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --name "Served By Description" \
 --type user-defined \
 --set enabled:true \
 --set attribute-type:description \
 --set base-dn:dc=example,dc=com \
 --set value:"Served by my favorite directory server" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=wlutz)" \
 description

dn: uid=wlutz,ou=People,dc=example,dc=com
```

```
description: Description on ou=People
description: Served by my favorite directory server
```

Collective Attributes

Collective attributes provide a standard mechanism for inheriting attribute values. DS servers support standard collective attributes, described in RFC 3671. The inherited values appear on all the entries in a subtree, optionally filtered by object class. Standard collective attribute type names have the prefix `c-`.

DS servers extend collective attributes to make them easier to use. You can define any DS attribute as collective with the `;collective` attribute option. Use LDAP filters in the subtree specification for fine-grained control over which entries inherit the values.

You establish an inheritance relationship between entries by specifying one of the following:

- Which attribute holds the DN of the entry to inherit attributes from.
- How to construct the RDN of the entry to inherit attributes from.

Class of Service

This example defines attributes that depend on the user's service level.

This example shows collective attributes that depend on a `classOfService` attribute value:

- For entries with `classOfService: bronze`, `mailQuota` is 1 GB, and `diskQuota` is 10 GB.
- For entries with `classOfService: silver`, `mailQuota` is 5 GB, and `diskQuota` is 50 GB.
- For entries with `classOfService: gold`, `mailQuota` is 10 GB, and `diskQuota` is 100 GB.

You define collective attributes in the user data using an LDAP subentry. As a result, collective attribute settings are replicated. Collective attributes use attributes defined in the directory schema. The following LDIF shows the schema definitions:

```
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( example-class-of-service-attribute-type
NAME 'classOfService'
```

```
EQUALITY caseIgnoreMatch
ORDERING caseIgnoreOrderingMatch
SUBSTR caseIgnoreSubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
SINGLE-VALUE
USAGE userApplications
X-ORIGIN 'DS Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-class-of-service-disk-quota
NAME 'diskQuota'
EQUALITY caseIgnoreMatch
ORDERING caseIgnoreOrderingMatch
SUBSTR caseIgnoreSubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
USAGE userApplications
X-ORIGIN 'DS Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-class-of-service-mail-quota
NAME 'mailQuota'
EQUALITY caseIgnoreMatch
ORDERING caseIgnoreOrderingMatch
SUBSTR caseIgnoreSubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
USAGE userApplications
X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( example-class-of-service-object-class
NAME 'cos'
SUP top
AUXILIARY
MAY ( classOfService $ diskQuota $ mailQuota )
X-ORIGIN 'DS Documentation Examples' )
```

The collective attribute definitions set the quotas depending on class of service:

```
dn: cn=Bronze Class of Service,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Bronze Class of Service
diskQuota;collective: 10 GB
```

```
mailQuota;collective: 1 GB
subtreeSpecification: { base "ou=People", specificationFilter "
(classOfService=bronze)" }

dn: cn=Silver Class of Service,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Silver Class of Service
diskQuota;collective: 50 GB
mailQuota;collective: 5 GB
subtreeSpecification: { base "ou=People", specificationFilter "
(classOfService=silver)" }

dn: cn=Gold Class of Service,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Gold Class of Service
diskQuota;collective: 100 GB
mailQuota;collective: 10 GB
subtreeSpecification: { base "ou=People", specificationFilter "
(classOfService=gold)" }
```

When the collective attributes are defined, you see the results on user entries. Before trying this example, set up a directory server with the `ds-evaluation` profile:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 classOfService mailQuota diskQuota

dn: uid=bjensen,ou=People,dc=example,dc=com
classOfService: bronze
mailQuota: 1 GB
diskQuota: 10 GB

$ ldapsearch \
```

```
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=kvaughan)" \
 classOfService mailQuota diskQuota

dn: uid=kvaughan,ou=People,dc=example,dc=com
classOfService: silver
mailQuota: 5 GB
diskQuota: 50 GB

$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=scarter)" \
 classOfService mailQuota diskQuota

dn: uid=scarter,ou=People,dc=example,dc=com
classOfService: gold
mailQuota: 10 GB
diskQuota: 100 GB
```

Inherit From the Manager

This example demonstrates how to set an employee's department number using the manager's department number.

The employee-manager relationship is defined by the employee's `manager` attribute. Each `manager` attribute specifies the DN of a manager's entry. The server looks up the manager's department number to set the attribute on the employee's entry.

The collective attribute subentry specifies that users inherit department number from their manager:

```
dn: cn=Inherit Department Number From Manager,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: inheritedCollectiveAttributeSubentry
```

```
objectClass: inheritedFromDNCollectiveAttributeSubentry
cn: Inherit Department Number From Manager
subtreeSpecification: { base "ou=People", specificationFilter "
(objectClass=posixAccount)" }
inheritFromDNAttribute: manager
inheritAttribute: departmentNumber
```

Babs Jensen's manager is Torrey Rigden:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
manager: uid=trigden, ou=People, dc=example,dc=com
```

Torrey's department number is 3001:

```
dn: uid=trigden,ou=People,dc=example,dc=com
departmentNumber: 3001
```

Babs inherits her department number from Torrey. Before trying this example, set up a directory server with the `ds-evaluation` profile:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 departmentNumber

dn: uid=bjensen,ou=People,dc=example,dc=com
departmentNumber: 3001
```

Inherit From the Locality

This example demonstrates how to set a user's default language preferences and street address based on locality.

For this example, the relationship between entries is based on locality. The collective attribute subentry specifies how to construct the RDN of the entry with the values to inherit:

```
dn: cn=Inherit From Locality,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: inheritedCollectiveAttributeSubentry
objectClass: inheritedFromRDNCollectiveAttributeSubentry
cn: Inherit From Locality
subtreeSpecification: { base "ou=People", specificationFilter "
(objectClass=posixAccount)" }
inheritFromBaseRDN: ou=Locations
inheritFromRDNAttribute: l
inheritFromRDNType: l
inheritAttribute: preferredLanguage
inheritAttribute: street
collectiveConflictBehavior: real-overrides-virtual
```

The RDN of the entry from which to inherit attributes is `l=localityName,ou=Locations`, where localityName is the value of the `l` (`localityName`) attribute on the user's entry.

In other words, if the user's entry has `l: Bristol`, then the RDN of the entry from which to inherit attributes starts with `l=Bristol,ou=Locations`. The actual entry looks like this:

```
dn: l=Bristol,ou=Locations,dc=example,dc=com
objectClass: top
objectClass: locality
objectClass: extensibleObject
l: Bristol
street: Broad Quay House, Prince Street
preferredLanguage: en-gb
```

The subentry specifies that the inherited attributes are preferred language and street address.

The object class `extensibleObject` allows the entry to take a preferred language. The object class `extensibleObject` means, "Let me add whatever attributes I want." The best practice is to add a custom auxiliary object class instead. The example uses shortcut `extensibleObject` for simplicity.

Notice the last line of the collective attribute subentry:

```
collectiveConflictBehavior: real-overrides-virtual
```

This line indicates that when a collective attribute clashes with a real attribute, the real value takes precedence over the virtual, collective value. You can set `collectiveConflictBehavior` to `virtual-overrides-real` for the opposite precedence, or to `merge-real-and-virtual` to keep both sets of values.

In this example, users can set their own language preferences. When users set language preferences manually, the user's settings override the locality-based setting.

Sam Carter is located in Bristol. Sam has specified no preferred languages:

```
dn: uid=scarter,ou=People,dc=example,dc=com
l: Bristol
```

Sam inherits the street address and preferred language from the Bristol locality. Before trying this example, set up a directory server with the `ds-evaluation` profile:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=scarter)" \
 preferredLanguage street

dn: uid=scarter,ou=People,dc=example,dc=com
preferredLanguage: en-gb
street: Broad Quay House, Prince Street
```

Babs's locality is San Francisco. Babs prefers English, but also knows Korean:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
preferredLanguage: en, ko;q=0.8
l: San Francisco
```

Babs inherits the street address from the San Francisco locality, but keeps her language preferences:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

```
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 preferredLanguage street

dn: uid=bjensen,ou=People,dc=example,dc=com
preferredLanguage: en, ko;q=0.8
street: 201 Mission Street Suite 2900
```

Inherit From a Parent Entry

This example demonstrates how to inherit a description from the parent entry.

The following collective attribute subentry specifies that entries inherit a description from their parent unless they already have a description:

```
dn: cn=Inherit Description From Parent,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: inheritedCollectiveAttributeSubentry
objectClass: inheritedFromDNCollectiveAttributeSubentry
cn: Inherit Description From Parent
subtreeSpecification: { base "", minimum 2, specificationFilter "
(objectClass=posixAccount)" }
inheritFromDNAttribute: entryDN
inheritFromDNParent: 1
inheritAttribute: description
collectiveConflictBehavior: real-overrides-virtual
```

In this example, `inheritFromDNAttribute` uses the virtual attribute `entryDN`. The setting `inheritFromDNParent: 1` indicates that the server should locate the parent entry by removing one leading RDN from the `entryDN`. (To inherit from the grandparent entry, you would specify `inheritFromDNParent: 2`, for example.)

Sam Carter's entry has no description attribute, and so inherits the parent's description:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=scarter)" \
```

```
 description

dn: uid=scarter,ou=People,dc=example,dc=com
description: Description on ou=People
description: Served by my favorite directory server
```

Babs Jensen's entry already has a description attribute, and so does not inherit from the parent:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 description

dn: uid=bjensen,ou=People,dc=example,dc=com
description: Original description
```

About Subentry Scope

LDAP subentries reside with the user data and so the server replicates them. Subentries hold operational data. They are not visible in search results unless explicitly requested. This section describes how a subentry's `subtreeSpecification` attribute defines the scope of the subtree that the subentry applies to.

An LDAP subentry's subtree specification identifies a subset of entries in a branch of the DIT. The subentry scope is these entries. In other words, these are the entries that the subentry affects.

The attribute value for a `subtreeSpecification` optionally includes the following parameters:

base
: Indicates the entry, *relative to the subentry's parent*, at the base of the subtree.

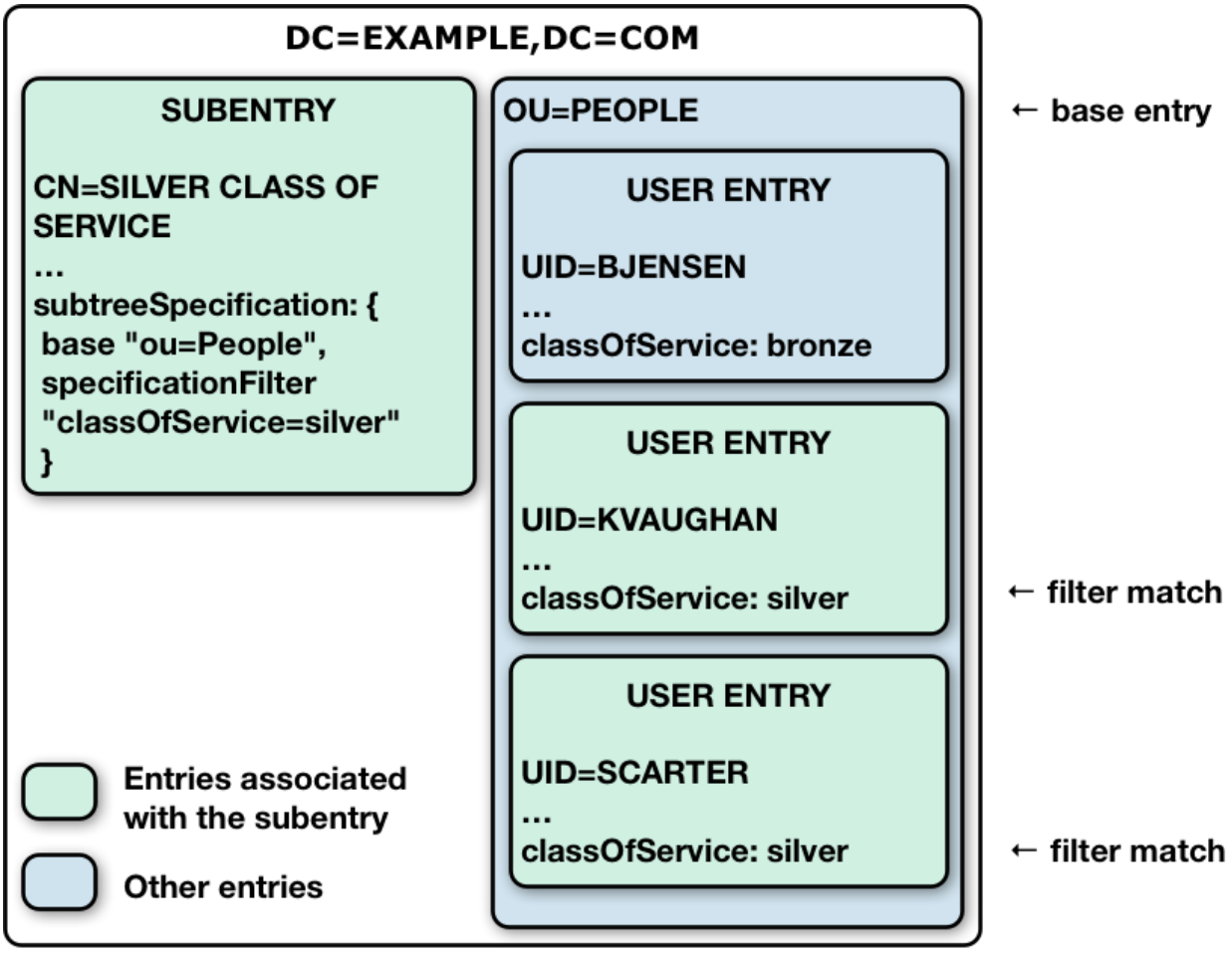
  By default, the base is the subentry's parent.

specificationFilter
: Indicates an LDAP filter. Entries matching the filter are in scope.

  DS servers extend the standard implementation to allow any search filter, not just an assertion about the `objectClass` attribute.

By default, all entries under the base entry are in scope.

The following illustration shows this for an example collective attribute subentry:



Notice that the base of `ou=People` on the subentry `cn=Silver Class of Service,dc=example,dc=com` indicates that the base entry is `ou=People,dc=example,dc=com`.

The filter `"(classOfService=silver)"` means that Kirsten Vaughan and Sam Carter's entries are in scope. Babs Jensen's entry, with `classOfService: bronze` does not match and is therefore not in scope. The `ou=People` organizational unit entry does not have a `classOfService` attribute, and so is not in scope, either.

Replication

Always-on directory services use replication to let applications continue reading and writing data, even when individual servers crash, and network connections are interrupted. DS replication is designed to make building highly available directory services straightforward.

About Replication

Replication is the process of copying updates between DS servers so all directory servers eventually converge on identical copies of directory data. DS servers that replicate their data are *replicas*. Since replication is *eventually convergent*, different replicas can be momentarily out of sync. If you lose an individual replica, or even an entire data center, the remaining replicas continue to provide service. Applications can still write changes to the directory data. Replication brings the replicas back in sync when the problem is repaired.

Replication uses a DS-specific protocol that works only between DS replicas. It replays update operations quickly, storing historical change data to resolve most conflicts automatically. For example, if two client applications separately update a user entry to change the phone number, replication can identify the latest change, and apply it on all replicas without human intervention. To prevent the historical change data from growing forever, DS replicas purge historical data that is older than a configurable interval (default: three days).

DS software supports replication over LAN and WAN networks. If you have multiple sites with many replicas communicating over the WAN, consider standalone replication servers. For details, see Install Standalone Servers.

Replication is resilient to host clock anomalies. You should, however, aim to keep server clocks synchronized using `ntpd`, for example. Keeping replica clocks synchronized helps prevent issues when validating certificates for secure connections, and makes it easier to compare timestamps from multiple replicas. Replication is designed to overcome the following issues:

- Clock skew between different replicas.

  Replication adjusts for skew automatically, and using `ntpd` further mitigates this.

  Very large skew, such as replicating with a system whose clock was started at 0 (January 1, 1970), has been seen to cause problems.

- Forward and backward clock adjustments on a single replica.

  Replication adjusts for these automatically.

  Very large changes, such as abruptly setting the clock back an entire day, have been seen to cause problems.

- Timezone differences and adjustments.

  Replication uses UTC time.

Very large host clock anomalies can result in the following symptoms:

- SSL certificate validation errors, when the clocks are far enough apart that the validity dates cannot be correctly compared.

- Problems with time-based settings in access control instruction subjects, and with features that depend on timestamps, such as password age and last login attributes.
- Misleading changelog timestamps and replication-related monitoring data.
- Incorrect replication conflict resolution.
- Incorrect replication purge delay calculation.

Ready to Replicate

When you set up a server, you can specify the following:

- The replication port.

  If specified, the setup process configures the server as a replication server.
- The bootstrap replication servers' host:port combinations.

  When the server starts, it contacts the bootstrap replication servers to discover other replicas and replication servers.

Setup profiles that create backends for schema and directory data configure replication domains for their base DNs. The server is ready to replicate that directory data when it starts up.

Replication initialization depends on the state of the data in the replicas.

DS replication shares changes, not data. When a replica applies an update, it sends a message to its replication server. The replication server forwards the update to all other replication servers, and to its replicas. The other replication servers do the same, so the update is eventually propagated to all replicas.

Each replica eventually converges on the same data by applying each update and resolving conflicts in the same way. As long as each replica starts from the same initial state, each replica eventually converges on the same state. It is crucial, therefore, for each replica to begin in the same initial state. Replicas cannot converge by following exactly the same steps from different initial states.

Internally, DS replicas store a shorthand form of the initial state called a generation ID. The generation ID is a hash of the first 1000 entries in a backend. If the replicas' generation IDs match, the servers can replicate data without user intervention. If the replicas' generation IDs do not match for a given backend, you must manually initialize replication between them to force the same initial state on all replicas.

If necessary, and before starting the servers, further restrict TLS protocols and cipher suites on all servers. This forces the server to use only the restricted set of protocols and cipher suites. For details, see TLS Settings.

Replication Per Base DN

The primary unit of replication is the *replication domain*. A replication domain has a base DN, such as `dc=example,dc=com`.

The set of DS replicas and replication servers sharing one or more replication domains is a *replication topology*. Replication among these servers applies to all the data under the domain's base DN.

The following example topology replicates `dc=example,dc=com`, `dc=example,dc=org`, and `dc=example,dc=net`. All the replication servers in a topology are fully connected, replicating the same data under each base DN:

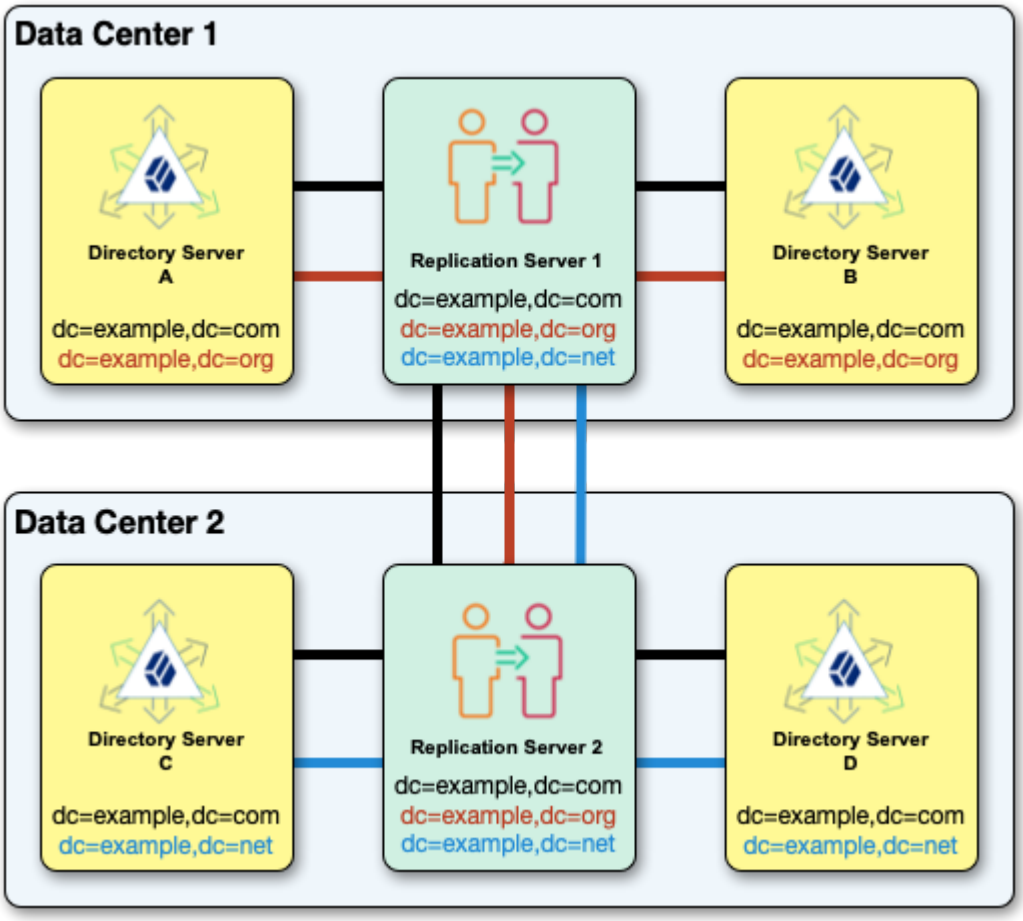


Figure 1. Correct replication configuration

Replication doesn't support separate, independent domains for the same base DN in the same topology. For example, you can't replicate two `dc=example,dc=org` domains with different data in the same topology:

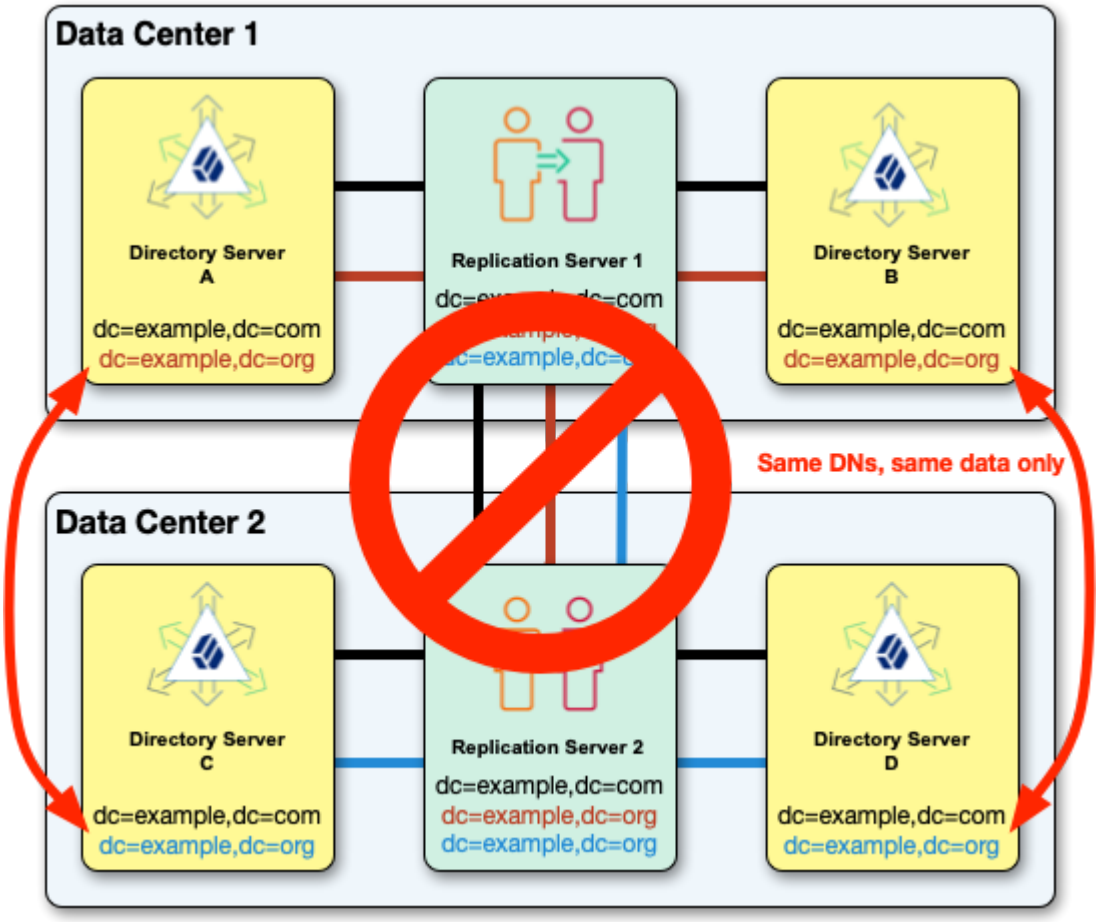


Figure 2. Incorrect replication configuration

When you set up a replication domain, replicate the data under the base DN among all the servers in the topology. If the data under a base DN is different on different servers, configure the servers appropriately:

Difference	Use this
Different data under the same base DN.	Separate replication topologies. In other words, put the replicas and replication servers in independent deployments unless the data matches.
Some replicas have only part of the data under a base DN.	Subtree Replication
Some replicas have only a subset of LDAP attributes.	Fractional Replication

Replication depends on the directory schema under `cn=schema`. If applications require specific, non-standard schema definitions or can update the LDAP schema online, replicate `cn=schema` with the other base DNs.

Port Use and Operations

DS servers listen on dedicated ports for administrative requests and for replication requests. These dedicated ports must remain open to remote requests from configuration tools and from other servers. *Make sure that firewall software allows connections to the administration and replication ports from all connecting DS servers.*

DS server configuration tools securely connect to administration ports. Administrative connections are short-lived.

DS replicas connect to DS replication ports for replication requests. A server listening on a replication port is called a *replication server*, whether it is running inside the same process as a directory server, or on a separate host system. Replication connections are long-lived. Each DS replica connects to a replication server upon initialization and then at startup time. The replica keeps the connection open to push and receive updates, although it can connect to another replication server.

The command to initialize replication uses the administrative port, and initialization uses the replication port:

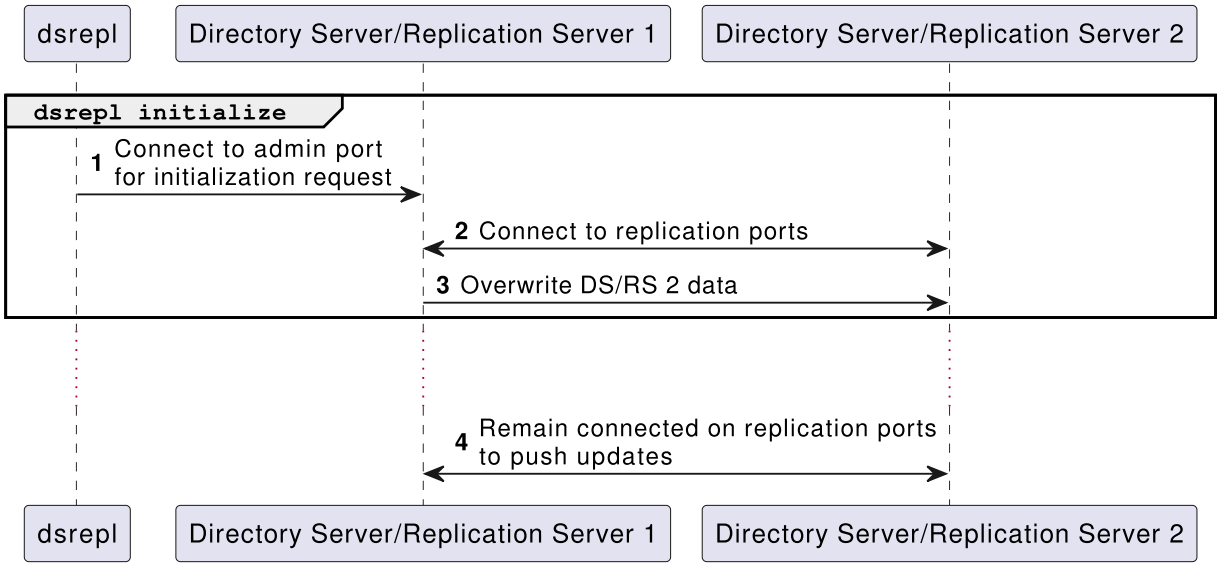


Figure 3. Manual Replication Initialization

DS replicas push updates to and receive updates from replication servers over replication connections. When processing an update, a replica (DS) pushes it to the replication server (RS) it is connected to. The replication server pushes the update to connected replicas and to other replication servers. Replicas always connect through replication servers.

A replica with a replication port and a changelog plays both roles (DS/RS), normally connecting to its own replication server. A standalone replica (DS) connects to a remote replication server (RS). The replication servers connect to each other. The following figure shows the flow of messages between standalone replicas and replication servers.

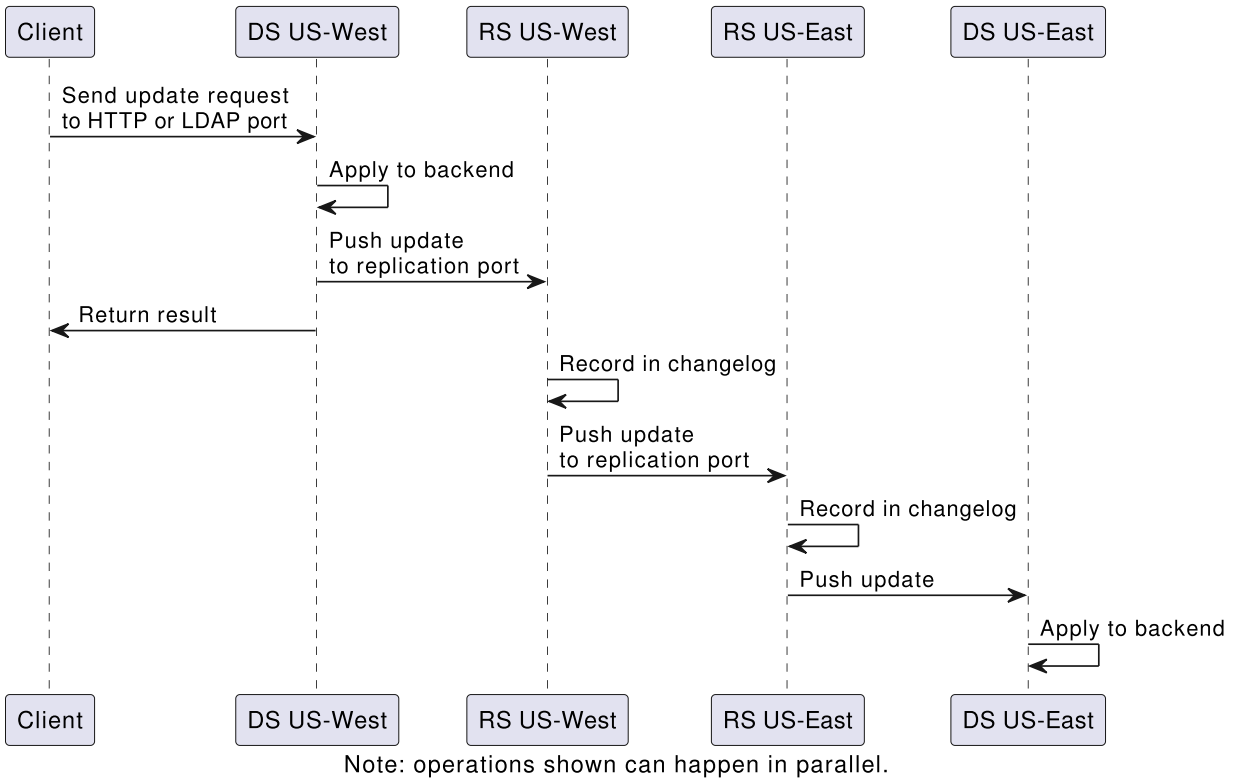
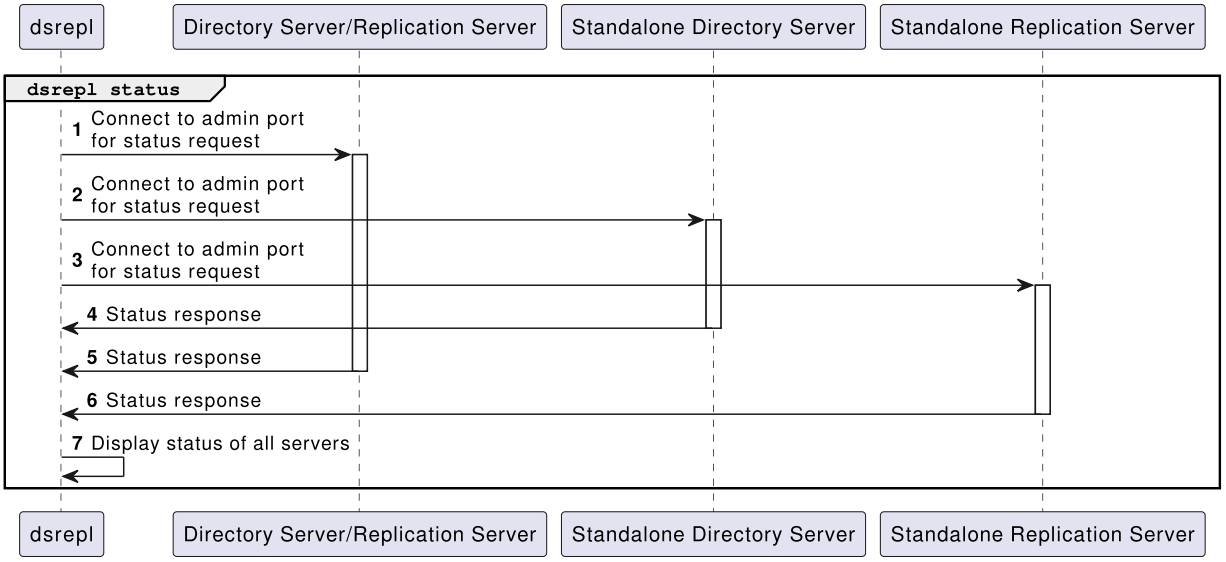


Figure 4. Data Replication

The command to monitor replication status uses the administration ports on multiple servers to connect and read monitoring information, as shown in the following sequence diagram:



Replication Connection Selection

DS servers can provide both directory services and replication services. The two services are not the same, even if they can run alongside each other in the same DS server in the same JVM.

Replication relies on the replication service provided by DS replication servers. DS directory servers (replicas) publish changes made to their data, and subscribe to

changes published by other replicas. The replication service manages replication data only, sending and receiving replication messages. A replication server receives, sends, and stores only changes to directory data, not the data itself.

The directory service manages directory data. It responds to requests, and stores directory data and historical information. For each replicated base DN, such as `dc=example,dc=com` or `cn=schema`, the directory service publishes changes to and subscribes to changes from a replication service. The directory service resolves any conflicts that arise when reconciling changes from other replicas, using the historical information about changes to resolve the conflicts. (Conflict resolution is the responsibility of the directory server rather than the replication server.)

After a directory server connects to a replication topology, it connects to one replication server at a time for a given domain. The replication server provides the directory server with the list of all replication servers for that base DN. Given this list, the directory server selects its preferred replication server when starting up, when it loses the current connection, or when the connection becomes unresponsive.

For each replicated base DN, a directory server prefers to connect to a replication server:

1. In the same JVM as the directory server.
2. In the same group as the directory server.

   By default, if no replication server in the same group is available, the directory server chooses a replication server from any available group.

   To define the order of failover across replication groups, set the global configuration property, group-id-failover-order. When this property is set and no replication server is available in the directory server's group, the directory server chooses a replication server from the next group in the list.
3. With the same initial data under the base DN as the directory server.
4. If initial data was the same, a replication server with all the latest changes from the directory server.
5. With the most available capacity relative to other eligible replication servers.

   Available capacity depends on how many replicas in the topology are already connected to the replication server, and what proportion of all replicas ought to be connected to the replication server.

   To determine what proportion ought to be connected, a directory server uses replication server weight. When configuring a replication server, you can assign it a weight (default: 1). The weight property takes an integer that indicates capacity relative to other replication servers. For example, a weight of 2 indicates a replication server that can handle twice as many connected replicas as one with weight 1.

The proportion that ought to be connected is `(replication server weight)/(sum of replication server weights)`. If there are four replication servers with weight 1, the proportion for each is 1/4.

Consider a `dc=example,dc=com` topology with five directory servers connected to replication servers A, B, and C, where:

- Two directory servers are connected to replication server A.
- Two directory servers are connected to replication server B.
- One directory server is connected to replication server C.

Replication server C is the server with the most available capacity. All other criteria being equal, replication server C is the server to connect to when another directory server joins the topology.

The directory server regularly updates the list of replication servers in case it must reconnect. As available capacity can change dynamically, a directory server can reconnect to another replication server to balance the replication load in the topology. For this reason, the server can also end up connected to different replication servers for different base DNs.

Manual Initialization

Manual initialization is not always required. Replication can proceed automatically when replicas start from the same initial data.

If this is not the case, manually initialize replication. How you initialize replication depends on your situation:

Initialization Options

Use Cases	Recommendations
Replicas installed with same data	Nothing to do (no manual initialization required)
Evaluating DS software Developing a directory solution	Initialize Over the Network *Limitations:* • Transmits all data over the network, so requires ample bandwidth; can be a problem for WAN links. • Rebuilds indexes and consumes significant system resources; can impact service performance.

Use Cases	Recommendations
New directory service, medium to large data set (> 500,000 entries)	Initialize From LDIF *Limitations:* • Rebuilds indexes and consumes significant system resources; can impact service performance.
Existing directory service, medium to large data set (> 500,000 entries)	Initialize From Backup *Limitations:* • All DS servers must be the same version. Backups are not guaranteed to be compatible across major and minor server releases.
New backend	Create a Backend, then one of: • Initialize Over the Network • Initialize From LDIF • Back Up the new backend, then Initialize From Backup *Limitations:* • The limitations depend on how you initialize the new backend, and are described above.
Broken data set	Disaster recovery *Limitations:* • This method **permanently loses recent changes**. • Applications using the changelog must be reinitialized after you restore the data.

Initialize Over the Network

Review Initialization Options before following these steps:

> 1. Manually initialize replication using the replication protocol's total update capability in one of these ways:

a. Overwrite the data in all replicas with the data from the replica where the command runs:

```
$ dsrepl \
 initialize \
 --baseDN dc=example,dc=com \
 --toAllServers \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --trustStorePath /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

b. Initialize a single other replica, identified by its server ID, from the replica where the command runs:

```
$ dsrepl \
 initialize \
 --baseDN dc=example,dc=com \
 --toServer ds-1 \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --trustStorePath /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

Initialize From LDIF

IMPORTANT

If you aim to return to a previous state of the data, or to initialize replicas with LDIF from a non-replicated environment, refer to Disaster recovery.

Review Initialization Options before following these steps:

Initialize each replica with the same LDIF:

1. Stop the server.

2. If desired, enable data confidentiality.

   For details, see Data Encryption and Encrypt External Changelog Data.

3. Import the LDIF.

   For details, see Import LDIF.

4. Start the server.

Initialize From Backup

Review Initialization Options before following these steps:

1. Stop the replica.
2. Restore the backend from backup.

   For details, see Restore.

3. Start the replica.

   Replication replays changes from other replicas that have happened since the backup was created.

Replication Status

The following command displays a snapshot of replication monitoring information:

```
$ dsrepl \
 status \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=monitor \
 --bindPassword password \
 --trustStorePath /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

Base DN            Status  Receive     Replay
                           delay (ms)  delay (ms)
-------------------------------------------------
dc=example,dc=com  OK         <delay>     <delay>
uid=monitor        OK         <delay>     <delay>
cn=schema          OK         <delay>     <delay>
```

The command connects to each known server to read status information. It will eventually time out if other servers cannot be contacted.

To get a balanced view of replication delays, monitor them over time. You can do this with repeated use of the **`dsrepl status`** command, or by reading the monitoring information over LDAP or HTTP. For details, see Replication Delay (Prometheus) or Replication Delay (LDAP).

Manual Purge

Over time, DS servers purge any historical data and changelog data older than the replication purge delay. To remove stale historical data, you can optionally trigger a purge task manually:

```
$ dsrepl \
 purge-meta-data \
 --baseDN dc=example,dc=com \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --trustStorePath /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

By default, this task runs for a maximum of one hour. If desired, set the maximum duration in seconds with the `--maximumDuration` option.

With earlier DS versions, you had to purge replicas from other servers' configurations after they were removed. DS servers do this automatically now. No administrative action is required.

Replication Groups

Define replication groups so that replicas connect first to local replication servers, only going outside the group when no local replication servers are available.

> For each group, set the appropriate group ID on the replication servers and the replicas. The following steps set up two replication groups, each with a replication server and a directory server. In a full-scale deployment, you would have multiple servers of each type in each group. The replicas and replication servers in the same location would belong to the same group:
>
> 1. Pick a group ID for each group.

The default group ID is `default`. For mixed topologies with replicas running older DS versions that support only numeric group IDs, this is equivalent to `1`.

2. Set the group ID for each group on the directory servers:

```
$ dsconfig \
 set-global-configuration-prop \
 --set group-id:US-East \
 --hostname ds.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 set-global-configuration-prop \
 --set group-id:US-West \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. Set the group ID for each group on the replication servers:

```
$ dsconfig \
 set-global-configuration-prop \
 --set group-id:US-East \
 --hostname rs.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 set-global-configuration-prop \
```

```
 --set group-id:US-West \
 --hostname rs2.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

Subtree Replication

To configure subtree replication, split the data across multiple servers.

For example, use different servers for `ou=People,dc=example,dc=com`, and other `dc=example,dc=com` data. Set up one set of servers with everything in `dc=example,dc=com` except `ou=People,dc=example,dc=com`. Set up a separate set of servers for `ou=People,dc=example,dc=com`. The former replicate `dc=example,dc=com`, the latter `ou=People,dc=example,dc=com`. For details, see Split Data.

Due to limitations described in Replication Per Base DN, the same servers cannot replicate both `dc=example,dc=com` and `ou=People,dc=example,dc=com` separately.

Fractional Replication

With fractional replication, you specify the attributes to include and to exclude using `fractional-include` and `fractional-exclude` configuration properties. Fractional replicas must respect LDAP schemas. Attributes that are required by the relevant object classes are included whether you specify them or not. Excluded attributes must be optional attributes of the relevant object classes.

Each attribute must remain on at least one replica. When you configure a replica to exclude an attribute, the replica checks that the attribute is never added to the replica as part of any LDAP operation. If you exclude the attribute everywhere, it can never be added anywhere.

When using fractional replication, initialize replication from LDIF. The import process imports only the data allowed by fractional replication. Be aware that you cannot create a replica with a full data set from a replica with only a subset of the data.

Replication servers filter objects for fractional replication. If you must prevent data from being replicated across a national boundary, for example, keep standalone replication servers in locations where you can store full entries and their changes. Outside that location, set up standalone replicas that receive the fractional entries.

The following example configures a fractional replica with a subset of `inetOrgPerson` attributes:

```
$ dsconfig \
 set-replication-domain-prop \
 --provider-name "Multimaster Synchronization" \
 --domain-name "dc=example,dc=com" \
 --set fractional-
include:inetorgperson:cn,givenname,mail,mobile,sn,telephonenumber
\
 --hostname replica.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

The following example excludes a custom attribute, `sessionToken` , on the replica:

```
$ dsconfig \
 set-replication-domain-prop \
 --provider-name "Multimaster Synchronization" \
 --domain-name "dc=example,dc=com" \
 --set fractional-exclude:*:sessionToken \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

This example only applies if you have defined a `sessionToken` attribute in the LDAP schema.

Read-Only Replicas

By default, all directory servers in a replication topology are read-write.

The following command causes the replica to accept only replication updates, and to refuse updates from client applications:

```
$ dsconfig \
 set-global-configuration-prop \
 --set writability-mode:internal-only \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

The following command resets the replica to the default behavior:

```
$ dsconfig \
 set-global-configuration-prop \
 --set writability-mode:enabled \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Trusted Replicas

By default, all directory servers in a replication topology trust all replicas. If a replica allows an update, then other servers relay and replay the update without further verification. This simplifies deployments where you control all the replicas.

In deployments where you do not control all the replicas, you can configure replication servers to accept updates only from trusted replicas. The trust depends on the certificate that a replica presents to the replication server when connecting. Specifically, replication servers can verify trust when:

- Trusted certificates have the OID `1.3.6.1.4.1.36733.2.1.10.1` in their extended key usage certificate extension.

  Use this policy when you control the CA signing the replicas' certificates, and can enforce that only authorized replicas have certificates with this setting.

- They store fingerprints for all trusted certificates in their configurations.

  Use this policy when you do not control the CA.

You choose the policy by setting the replication server advanced property, allow-updates-policy. It takes the following values:

all
: (Default) Trust all replicas.

verify-certificate-key-usage
: Only trust updates from replicas whose certificates' `ExtendedKeyUsage` includes `1.3.6.1.4.1.36733.2.1.10.1`.

verify-certificate-fingerprint
: Only trust updates from replicas with certificate fingerprints in the advanced property, allow-updates-server-fingerprints.

If a replication server does not trust an update, it logs an error message explaining why. The update is not replicated, and therefore may cause replication to diverge on the untrusted replica. Configure the untrusted replicas you control to be read-only, as described in Read-Only Replicas.

Trust Extended Key Usage

1. For each trusted DS server, get the certificate for replication signed with the appropriate extended key usage.

   The following example demonstrates a certificate signing request with the appropriate extended key usage:

   ```
   $ keytool \
    -certreq \
    -ext
   ExtendedKeyUsage:critical=clientAuth,serverAuth,1.3.6.1.4.
   1.36733.2.1.10.1 \
    -alias ssl-key-pair \
    -keystore /path/to/opendj/config/keystore \
    -storepass:file /path/to/opendj/config/keystore.pin \
    -file ssl-key-pair.csr
   ```

   The full process for each server involves generating a certificate signing request, signing the certificate in the request with the CA signing certificate, and importing the CA-signed certificate on the server.

2. For each replication server, update the configuration to trust certificates with the appropriate extended key usage:

```
$ dsconfig \
 set-replication-server-prop \
 --provider-name "Multimaster Synchronization" \
 --set allow-updates-policy:verify-certificate-key-usage \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

At this point, the replication server can trust the other server's updates.

Trust Fingerprints

1. For each trusted DS server, get the certificate fingerprint:

```
$ keytool \
 -list \
 -alias ssl-key-pair \
 -keystore /path/to/opendj/config/keystore \
 -storepass:file /path/to/opendj/config/keystore.pin \
ssl-key-pair, <date>, PrivateKeyEntry,
Certificate fingerprint (SHA-256):
05:55:BD:A5:E1:4C:35:A6:A5:4E:78:DD:3E:FD:EA:5A:66:5D:E0:D
C:9C:C5:18:7E:E9:CA:A9:1E:CD:87:4B:78
```

2. For each replication server, update the configuration to trust replicas by their certificate fingerprints:

```
$ dsconfig \
 set-replication-server-prop \
 --provider-name "Multimaster Synchronization" \
 --set allow-updates-policy:verify-certificate-fingerprint
\
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

```
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. For each replication server, update the configuration to recognize each certificate fingerprint.

   The following example demonstrates adding a trusted certificate fingerprint:

```
$ dsconfig \
 set-replication-server-prop \
 --provider-name "Multimaster Synchronization" \
 --add allow-updates-server-fingerprints:\
"{SHA-
256}05:55:BD:A5:E1:4C:35:A6:A5:4E:78:DD:3E:FD:EA:5A:66:5D:
E0:DC:9C:C5:18:7E:E9:CA:A9:1E:CD:87:4B:78" \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

   At this point, the replication server can trust the other server's updates.

4. Repeat the relevant steps each time a trusted certificate changes.

Listen Addresses

When configuring a server on a multi-homed system with multiple IP addresses, you can specify the listen addresses. By default, the replication server listens on all network interfaces.

The following example configures the server to listen only on `192.168.0.10` for all connections:

```
$ dsconfig \
 set-global-configuration-prop \
 --set advertised-listen-address:192.168.0.10 \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
```

```
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--no-prompt
```

For details, see advertised-listen-address.

Disk Space Thresholds

Replication servers record changes in changelog database files under the `opendj/changelogDb` directory.

WARNING

Do not compress, tamper with, or otherwise alter changelog database files directly, unless specifically instructed to do so by a qualified ForgeRock technical support engineer.

External changes to changelog database files can render them unusable by the server.

Replication server configuration objects have changelog database properties, `disk-low-threshold`, and `disk-full-threshold`:

- When available disk space falls below `disk-low-threshold`, the replication server triggers a warning alert notification to let you know to free disk space.
- When available disk space falls below `disk-full-threshold`, the replication server triggers another warning alert notification, and *disconnects from the replication topology*.

  Connected directory servers fail over to another replication server until available disk space is above the `disk-full-threshold` again.

Set the thresholds high enough to allow time to react after the initial alert.

The following example sets `disk-low-threshold` to 10 GB and `disk-full-threshold` to 5 GB:

```
$ dsconfig \
 set-replication-server-prop \
 --provider-name "Multimaster Synchronization" \
 --set "disk-low-threshold:10 GB" \
 --set "disk-full-threshold:5 GB" \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
```

```
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--no-prompt
```

The disk-low-threshold and disk-full-threshold properties are *advanced* properties. Examine their values with the **`dsconfig --advanced`** option.

Recover From User Error

It is possible to restore accidentally deleted or changed data.

Changes to a replicated DS directory service are similar to those made with the Unix **`rm`** command, but with a twist. With the **`rm`** command, if you make a mistake you can restore your files from backup, and lose only the work done since the last backup. If you make a mistake with an update to the directory service, after you restore from backup, replication efficiently replays your mistake over the data you restored.

There is more than one way to recover from user error. None of the ways are limited to changing DS settings. All involve manually fixing mistakes.

Consider these alternatives:

- Encourage client applications to provide end users with undo capability.

  In this case, client applications take responsibility for maintaining an undo history.

- Maintain a record of each update to the service, so that you can manually "undo" mistakes.

  You can use the external changelog. The external changelog is enabled with replication, and does not use additional space.

  For instructions, see Changelog for Notifications. In particular, see Include Unchanged Attributes on saving what is deleted as well as changed.

  DS servers can write to a file-based audit log. The audit log does not help with a general solution in this case. The DS audit log records only changes to the data. When you delete an entry, the audit log does not record the entry before deletion. The following example shows audit log records for changes made to Barbara Jensen's entry:

  ```
  # <datestamp>; conn=<number>; op=<number>
  dn: uid=bjensen,ou=People,dc=example,dc=com
  changetype: modify
  replace: description
  description: This is the description I want.
  -
  ```

```
replace: modifiersName
modifiersName: uid=admin
-
replace: modifyTimestamp
modifyTimestamp: <timestamp>

# <datestamp>; conn=<number>; op=<number>
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: I never should have changed this!
-
replace: modifiersName
modifiersName: uid=admin
-
replace: modifyTimestamp
modifyTimestamp: <timestamp>

# <datestamp>; conn=<number>; op=<number>
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: delete
```

You can use these records to fix the mistaken update to the description. However, the audit log lacks the information needed to restore Barbara Jensen's deleted entry.

- For administrative errors that involve directory data, use the external changelog if possible.

  If not, an alternative technique consists of restoring backup to a new server set up to not replicate. (Replication replays all updates, including mistakes.) Compare data on the separate restored server to the live replicas, and fix the mistakes manually.

  A more drastic alternative is to rebuild the entire service from backup, as described in Disaster recovery. This alternative is only recommended in the case of a major error where you have a very fresh backup (taken immediately before the error), and no client applications are affected.

- For administrative configuration errors that prevent servers from starting, know that DS servers keep snapshots of the main configuration file, including the `opendj/var/config.ldif.startok` file, and the files in the `opendj/var/archived-configs/` directory.

  Compare the current configuration with the earlier configurations, stop the server, and repair mistakes manually. Take care to avoid trailing white space at the end of LDIF lines.

Replication Conflicts

Replication is eventually consistent by design to support basic write availability. Changes are applied locally and then replayed to remote replicas. This means it is possible to have conflicts. A *replication conflict* arises when incompatible changes are made concurrently to multiple read-write replicas.

Two types of conflicts happen: *modify conflicts* and *naming conflicts*. Modify conflicts involve concurrent modifications to the same entry. Naming conflicts involve other operations that affect the DN of the entry.

Replication resolves modify conflicts, and many naming conflicts by replaying the changes in the correct order. To determine the relative order in which changes occurred, replicas retain historical information for each update. This information is stored in the target entry's `ds-sync-hist` operational attribute.

Replication resolves these conflicts automatically using the historical information to order changes correctly:

- The attributes of a given entry are modified concurrently in different ways on different replicas.
- An entry is renamed on one replica while being modified on another replica.
- An entry is renamed on one replica while being renamed in a different way on another replica.
- An entry is deleted on one replica while being modified on another replica.
- An entry is deleted and another entry with the same DN added on one replica while the same entry is being modified on another replica.

Replication cannot resolve these particular naming conflicts. You must resolve them manually:

- Different entries with the same DN are added concurrently on multiple replicas.
- An entry on one replica is moved (renamed) to use the same DN as a new entry concurrently added on another replica.
- A parent entry is deleted on one replica, while a child entry is added or renamed concurrently on another replica.

When replication cannot resolve naming conflicts automatically, the server renames the conflicting entry using its `entryUUID` operational attribute. The resulting conflicting entry has a DN with the following form:

```
entryuuid=entryUUID-value+original-RDN,original-parent-DN
```

For each conflicting entry named in this way, resolve the conflict manually:

1. Get the conflicting entry or entries, and the original entry if available.

   The following example shows the result on one replica of a naming conflict when a `newuser` entry was added concurrently on two replicas:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --baseDN dc=example,dc=com \
 "(uid=newuser)"

dn: uid=newuser,ou=People,dc=example,dc=com
objectClass: top
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
mail: newuser@example.com
sn: User
cn: New User
ou: People
description: Added on server 1
uid: newuser

dn: entryuuid=2f1b58c3-4bee-4215-88bc-
88202a7bcb9d+uid=newuser,ou=People,dc=example,dc=com
objectClass: top
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
mail: newuser@example.com
sn: User
cn: New User
ou: People
description: Added on server 2
uid: newuser
```

2. To preserve changes made on the conflicting entry or entries, apply the changes manually.

   The following example shows a modification to preserve both description values:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --bindDn uid=admin \
 --bindPassword password << EOF
dn: uid=newuser,ou=People,dc=example,dc=com
changetype: modify
add: description
description: Added on server 2
EOF
```

For additional examples demonstrating how to apply changes to directory entries, see LDAP Updates.

3. After making any necessary changes, manually delete the conflicting entry or entries.

   The following example deletes the conflicting entry:

```
$ ldapdelete \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --bindDN uid=admin \
 --bindPassword password \
 entryuuid=2f1b58c3-4bee-4215-88bc-
88202a7bcb9d+uid=newuser,ou=People,dc=example,dc=com
```

   For additional examples, see Delete Entries.

Bootstrap Replication Servers

A *bootstrap replication server* is one of the replication servers in a deployment that a server should contact to discover all the other servers in the deployment.

Add a Bootstrap Replication Server

After you add a replication server to a deployment, add it to the other servers' bootstrap-replication-server settings.

Apply these steps for each server whose configuration references the new replication server to add:

1. Add the bootstrap replication server to the server's configuration:

```
$ dsconfig \
 set-synchronization-provider-prop \
 --provider-name "Multimaster Synchronization" \
 --add bootstrap-replication-server:new-
rs.example.com:8989 \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

2. Restart the server for the changes to take effect.

Remove a Bootstrap Replication Server

After you remove a replication server from a deployment, remove it from other servers' `bootstrap-replication-server` settings.

Apply these steps for each server whose configuration references the replication server that you removed:

1. Remove the bootstrap replication server from the server's configuration:

```
$ dsconfig \
 set-synchronization-provider-prop \
 --provider-name "Multimaster Synchronization" \
 --remove bootstrap-replication-server:removed-
rs.example.com:8989 \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
```

```
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

2. Restart the server for the changes to take effect.

Disable Replication

Disable replication temporarily

WARNING

Do not allow modifications on the replica for which replication is temporarily stopped. No record of such changes is kept, and the changes cause replication to diverge.

Follow these steps to disable replication temporarily for a replica and drop any changes that occur:

1. Prevent changes to the affected data.

   For details, see Read-Only Replicas.

2. Disable the replication mechanism:

```
$ dsconfig \
 set-synchronization-provider-prop \
 --provider-name "Multimaster Synchronization" \
 --set enabled:false \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. Perform whatever operations are required.

4. Enable the replication mechanism again:

```
$ dsconfig \
 set-synchronization-provider-prop \
```

```
 --provider-name "Multimaster Synchronization" \
 --set enabled:true \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

5. Allow changes to the affected data.

   For details, see Read-Only Replicas.

Before removing a server from a group of replicated servers, disable replication as described. When the server you remove is a bootstrap replication server, also remove it from the configuration on all other servers.

Stop replicating permanently

You might remove a server from a replication topology because:

- The DS server is no longer needed.

  For example, you are scaling a deployment down, or retiring an old server that you replaced with a newer one.

- Someone configured replication between DS servers that should be independent.

  For example, at setup time, replication was configured to include all six replicas in three data centers, but the expected configuration was three separate directory service deployments with two replicas in each data center.

  In this case, you must permanently change which servers replicate with each other.

NOTE

The steps that follow only apply to deployments of DS 7 and later servers.

If you are upgrading from older servers and have a mix of DS 7 and earlier servers, see the Upgrade documentation instead.

To remove a server that is no longer needed:

1. Uninstall the server.

   For details, see Uninstallation.

2. If the server is referenced in other servers' `bootstrap-replication-server` settings, remove it.

   For details, see Remove a Bootstrap Replication Server.

3. The automated purge process eventually removes historical data and changelog data for old servers.

   You can optionally trigger a purge task manually, as described in Manual Purge.

To change which servers replicate with each other:

1. Prevent changes to the affected data.

   For details, see Read-Only Replicas.

2. Perform these steps in parallel on all affected servers:
   a. Disable the replication mechanism.

      For details, see Disable replication temporarily.

   b. Adjust the `bootstrap-replication-server` settings to limit replication as desired.

   c. Enable the replication mechanism again.

   d. Restart the server for the changes to take effect.

3. Allow changes to the affected data.

4. Delete entries that were erroneously replicated.

   For details, see Delete Entries.

5. The automated purge process eventually removes historical data and changelog data for old servers.

   You can optionally trigger a purge task manually, as described in Manual Purge.

Changelog for Notifications

Some applications require notification when directory data updates occur. For example, an application might need to sync directory data with another database, or the application might need to kick off other processing when certain updates occur. DS replication servers provide an external changelog that replications can use to read changes. This mechanism is more scalable and robust than LDAP persistent searches.

By default, changelog database files are found under the `opendj/changelogDb` directory.

WARNING

WARNING

Do not compress, tamper with, or otherwise alter changelog database files directly, unless specifically instructed to do so by a qualified ForgeRock technical support engineer.

External changes to changelog database files can render them unusable by the server.

Enable the External Changelog

DS servers that have a replication server port and an LDAP port publish an external changelog over LDAP:

Ports Configured	Examples	Notes
LDAP or LDAPS port	`1389` , `1636`	LDAP client applications use the LDAP or LDAPS port to read changelog data. A standalone replication server may not have an LDAP or LDAPS port configured.
Replication port	`8989`	Servers with replication ports maintain a changelog for their own use. The changelog is exposed over LDAP under the base DN, `cn=changelog` . Standalone directory servers do not maintain a changelog by default.

1. Make sure an LDAP or LDAPS port is configured so that LDAP client applications can read the changelog.
2. Depending on the server configuration, enable the changelog one of the following ways:
    a. For servers with replication ports, make sure replication is properly configured.

    The following example shows how the directory superuser can read the changelog:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore
```

```
\
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN cn=changelog \
 --searchScope base \
 "(&)"

dn: cn=changelog
objectclass: top
objectclass: container
cn: changelog
```

b. For standalone directory servers without replication ports, you can manually enable the changelog. These steps cause the server to begin maintaining a replication changelog, which consumes disk space:

- Update the default replication synchronization configuration entry as in the following example:

```
$ dsconfig \
 set-synchronization-provider-prop \
 --provider-name "Multimaster Synchronization" \
 --set bootstrap-replication-server:localhost:8989
\
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore
/path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

- Create a replication server configuration entry as in the following example:

```
$ dsconfig \
 create-replication-server \
 --provider-name "Multimaster Synchronization" \
 --set replication-port:8989 \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
```

```
 --bindPassword password \
 --usePkcs12TrustStore
/path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

- Create a replication domain configuration entry as in the following example:

```
$ dsconfig \
 create-replication-domain \
 --provider-name "Multimaster Synchronization" \
 --domain-name "Example.com" \
 --set base-dn:dc=example,dc=com \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore
/path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. Make sure the user who needs to read changelog data has the `changelog-read` privilege, and has access to read entries under `cn=changelog`.

   For details, see Let a User Read the Changelog.

Encrypt changelog data

DS servers do not encrypt changelog data on disk by default. Any user with system access to read directory files can potentially read external changelog data.

In addition to preventing read access by other users, you can configure confidentiality for changelog data. When confidentiality is enabled, the server encrypts changelog records before storing them on disk. The server decrypts changelog records before returning them to client applications.

IMPORTANT

Encrypting stored directory data does not prevent it from being sent over the network in the clear.

Use secure connections to protect data sent over the network.

DS servers encrypt data using symmetric keys. Servers store symmetric keys, encrypted with the shared master key, with the data they encrypt. As long as servers have the same shared master key, any server can recover symmetric keys needed to decrypt data.

Symmetric keys for (deprecated) reversible password storage schemes are the exception to this rule. When you configure a reversible password storage scheme, enable the `adminRoot` backend, and configure a replication domain for `cn=admin data`.

Encrypting and decrypting data require cryptographic processing that reduces throughput, and extra space for larger encrypted values. Tests with default settings show that the cost of enabling confidentiality can be quite modest. Your results can vary based on the host system hardware, the JVM, and the settings for `cipher-transformation` and `cipher-key-length`. Make sure you test your deployment to qualify the impact of confidentiality before changing settings in production.

Follow these steps to enable confidentiality:

1. Before you enable confidentiality on a replication server for the changelog data, first enable confidentiality for data stored in directory backends.

   For details, see Data Encryption.

2. Enable changelog confidentiality with the default encryption settings:

   ```
   $ dsconfig \
    set-replication-server-prop \
    --provider-name "Multimaster Synchronization" \
    --set confidentiality-enabled:true \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePassword:file
   /path/to/opendj/config/keystore.pin \
    --no-prompt
   ```

   Encryption applies to the entire changelog regardless of the confidentiality settings for each domain.

   After confidentiality is enabled, new changelog records are encrypted. DS servers do not rewrite old records in encrypted form.

3. If necessary, adjust additional confidentiality settings.

Use the same cipher suite for changelog confidentiality and data confidentiality.

The default settings for confidentiality are `cipher-transformation: AES/GCM/NoPadding`, and `cipher-key-length: 128`. This means the algorithm is the Advanced Encryption Standard (AES), and the cipher mode is Galois/Counter Mode (GCM). The syntax for the `cipher-transformation` is *`algorithm/mode/padding`*. You must specify the *algorithm*, *mode*, and *padding*. When the algorithm does not require a mode, use `NONE`. When the algorithm does not require padding, use `NoPadding`.

Let a User Read the Changelog

For a user to read the changelog, the user must have access to read, search, and compare changelog attributes, might have access to use the control to read the external changelog, and must have the `changelog-read` privilege.

1. Give the user access to read and search the changelog.

   The following example adds two global ACIs. The first ACI gives `My App` read access to root DSE attributes that hold information about the changelog. The second ACI gives `My App` read access to the changelog data:

```
$ dsconfig \
 set-access-control-handler-prop \
 --add global-aci:"(target=\"ldap:///\")\
(targetattr=\"changeLog||firstChangeNumber||lastChangeNumb
er||lastExternalChangeLogCookie\")\
(version 3.0; acl \"My App can access changelog attributes
on root DSE\"; \
allow (read,search,compare) \
userdn=\"ldap:///cn=My App,ou=Apps,dc=example,dc=com\";)"
\
 --add global-aci:"(target=\"ldap:///cn=changelog\")
(targetattr=\"*||+\")\
(version 3.0; acl \"My App can access cn=changelog\"; \
allow (read,search,compare) \
userdn=\"ldap:///cn=My App,ou=Apps,dc=example,dc=com\";)"
\
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

```
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

The IDM liveSync feature requires access to the root DSE attributes `changeLog`, `firstChangeNumber`, and `lastChangeNumber`.

2. Give the user access to use the public changelog exchange control.

   The following example adds a global ACI to give `My App` access to use the control:

```
$ dsconfig \
 set-access-control-handler-prop \
 --add global-aci:"(targetcontrol=\"Ecl\")\
(version 3.0; acl \"My App control access\"; \
allow (read) \
userdn=\"ldap:///cn=My App,ou=Apps,dc=example,dc=com\";)"
\
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. Give the user the `changelog-read` privilege:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=My App,ou=Apps,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: changelog-read
EOF
```

4. Check that the user can read the changelog:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN cn=changelog \
 --control "ecl:false" \
 "(&)" \
 changes changeLogCookie targetDN
```

Include Unchanged Attributes

The changes returned from a search on the external changelog include only what was actually changed. If you have applications that need additional attributes published with every changelog entry, regardless of whether the attribute itself has changed, specify those with the `ecl-include` and `ecl-include-for-deletes` properties:

1. Set the attributes to include for all update operations:

```
$ dsconfig \
 set-replication-domain-prop \
 --provider-name "Multimaster Synchronization" \
 --domain-name dc=example,dc=com \
 --set ecl-include:"@person" \
 --set ecl-include:entryUUID \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

   The `entryUUID` can be useful, for example, to ensure integrity when entries are renamed.

2. Set the attributes to include for deletes:

```
$ dsconfig \
 set-replication-domain-prop \
 --provider-name "Multimaster Synchronization" \
 --domain-name dc=example,dc=com \
 --add ecl-include-for-deletes:"*" \
 --add ecl-include-for-deletes:"+" \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

With the default configuration, the changelog records only the DN of the deleted entry.

Exclude a Domain

Exclude domains to prevent applications that read the external changelog from having to process update notifications for entries that are not relevant to them:

1. Change the replication server configuration to exclude the domain by base DN.

   The following example prevents the changelog from sending notifications about Example.com entries:

```
$ dsconfig \
 set-replication-server-prop \
 --provider-name "Multimaster Synchronization" \
 --set changelog-enabled-excluded-
domains:dc=example,dc=com \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

Align Draft Change Numbers

The external changelog can be used by applications that follow the Internet-Draft: Definition of an Object Class to Hold LDAP Change Records, and cannot process changelog cookies. Only default change number indexing is required to get the objects specified for this format, but there are steps you must perform to align change numbers across servers.

Change numbers described in the Internet-Draft are simple numbers, not cookies. When changelog numbers are aligned, applications can fail over from one server to another when necessary.

If you do not align the change numbers, each server keeps its own count. The same change numbers can refer to different changes on different servers.

For example, if you manually initialize a server from another, the last change numbers are likely to differ. The following example shows different last change numbers for two such servers:

```
$ ldapsearch \
 --hostname existing-ds.example.com \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN "" \
 --searchScope base \
 "(&)" lastChangeNumber

dn:
lastChangeNumber: 285924

Result Code:  0 (Success)

$ ldapsearch \
 --hostname new-ds.example.com \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
```

```
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN "" \
 --searchScope base \
 "(&)" lastChangeNumber

dn:
lastChangeNumber: 198643

Result Code:  0 (Success)
```

Follow these steps to align the change numbers with those of an existing server:

1. Make sure that the new server has the same replication configuration as the existing server.

   The change number calculations must be the same on both servers. Specifically, both servers must replicate data under the same base DNs. If the base DN configurations differ, the change numbers cannot be aligned.

2. If you must start the new servers's change numbering from a specific change, determine the `changeNumber` to use.

   The `changeNumber` must be from a change that has not yet been purged following the replication purge delay (default: 3 days).

3. Reset the change number on the new server to the change number from the existing server.

   The following example does not specify the change number to use. By default, the new server uses the last change number from the existing server:

   ```
   $ dsrepl \
    reset-change-number \
    --sourceHostname existing-ds.example.com \
    --sourcePort 4444 \
    --sourceBindDn uid=admin \
    --sourceBindPasswordPassword password \
    --targetHostname new-ds.example.com \
    --targetPort 4444 \
    --targetBindDn uid=admin \
    --targetBindPasswordPassword password \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePassword:file
   /path/to/opendj/config/keystore.pin \
    --no-prompt
   ```

> The new server's changelog now starts with the last change number from the existing server. Earlier change numbers are no longer present in the new servers's changelog.

Disable Change Number Indexing

By default, a replication server indexes change numbers for replicated user data. This allows applications to get update notifications by change number. Indexing change numbers requires additional CPU, disk I/O, and storage. Disable it if none of your applications require change number-based browsing:

1. Adjust the replication server settings appropriately for your deployment:
    a. If applications need change notifications, but use changelog cookies rather than change numbers, set `changelog-enabled:enabled-cookie-mode-only` on each server.

    The replication server no longer indexes change numbers, and so has less work to do.

    b. If no applications need notifications, set `changelog-enabled:disabled` on each server after enabling replication.

    c. If applications need notifications for some domains but not for others, set `changelog-enabled-excluded-domains` on each server after enabling replication.

Referrals

Referrals point directory clients to another directory container, which can be another directory server running elsewhere, or another container on the same server. The client receiving a referral must use the other container to complete the request.

NOTE

Some clients follow referrals on your behalf by default. The DS commands do not follow referrals.

Referrals are used, for example, when directory data is temporarily unavailable due to maintenance. Referrals can also be used when a container holds only some of the directory data for a base DN, and points to other containers for branches whose data is not available locally.

Referrals are entries with https://tools.ietf.org/html/rfc4516 `ref` attribute values that point elsewhere. The `ref` attribute type is required by the `referral` object class. The

`referral` object class is structural. By default, you cannot add it to an entry that already has a structural object class defined. (When adding a `ref` attribute to an entry with an existing structural object class, use the `extensibleObject` auxiliary object class.)

DS servers return referrals to requests that target the affected entry or its child entries. Client applications must be capable of following the referral returned. When the directory responds with a referral, the client can construct new operations and try them again.

The Manage DSAIT control lets you access the referral entry, rather than get a referral. It has OID `2.16.840.1.113730.3.4.2`. The control is described in RFC 3296.

Manage Referrals

Suppose the entries below `ou=Subscribers,dc=example,dc=com` are stored on a different directory server at `referral.example.com:1636`. You can create a LDAP referral to reference the remote entries.

Before creating the LDAP referral, give directory administrators access to use the Manage DSAIT control, and to manage the `ref` attribute:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --add global-aci:"(targetcontrol=\"ManageDsaIt\")\
 (version 3.0; acl \"Allow Manage DSAIT control\"; allow(read)\
 groupdn=\"ldap:///cn=Directory
Administrators,ou=Groups,dc=example,dc=com\";)" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: dc=example,dc=com
changetype: modify
```

```
add: aci
aci: (target="ldap:///dc=example,dc=com")(targetattr="ref")
  (version 3.0; acl "Admins can manage referrals"; allow(all)
  (groupdn = "ldap:///cn=Directory
Administrators,ou=Groups,dc=example,dc=com");)
EOF
```

To create an LDAP referral, either create a referral entry, or add the `extensibleObject` object class and the `ref` attribute with an LDAP URL to an existing entry. The example above creates a referral entry at `ou=Subscribers,dc=example,dc=com`:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: ou=Subscribers,dc=example,dc=com
objectClass: top
objectClass: extensibleObject
objectClass: organizationalUnit
ou: Subscribers
ref:
ldaps://referral.example.com:1636/ou=Subscribers,dc=example,dc=com
EOF
```

DS servers can now return a referral for operations under `ou=Subscribers`:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN ou=subscribers,dc=example,dc=com \
 "(uid=*)"

# The LDAP search request failed: 10 (Referral)
# Additional Information:  A referral entry
ou=Subscribers,dc=example,dc=com indicates that the operation must
be processed at a different server
# Matched DN:  ou=Subscribers,dc=example,dc=com
```

To access the entry instead of the referral, use the Manage DSAIT control:

```
$ ldapsearch \
 --control 2.16.840.1.113730.3.4.2:true \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN ou=subscribers,dc=example,dc=com \
 "(&)" \
 ref

dn: ou=Subscribers,dc=example,dc=com
ref:
ldaps://referral.example.com:1636/ou=Subscribers,dc=example,dc=com
```

You can use the Manage DSAIT control to change the referral with the **`ldapmodify`** command:

```
$ ldapmodify \
 --control 2.16.840.1.113730.3.4.2:true \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: ou=Subscribers,dc=example,dc=com
changetype: modify
replace: ref
ref: ldap://localhost:1636/ou=People,dc=example,dc=com
EOF
```

Attribute Uniqueness

Some attribute values must remain unique. For example, if you use `uid` as the RDN for millions of user entries, you must avoid two or more identical `uid` values. As another

example, if credit card or mobile numbers are stored on directory attributes, you want to be certain that neither is shared with another customer.

DS servers use the unique attribute plugin to ensure attribute value uniqueness. In a deployment with multiple replicas and unique attributes, direct updates for each unique attribute to a single replica at a time as described below.

Enable Unique UIDs

By default, the unique attribute plugin is configured to ensure unique `uid` values:

1. Set the base DN where the `uid` should have unique values, and enable the plugin:

```
$ dsconfig \
 set-plugin-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --plugin-name "UID Unique Attribute" \
 --set base-dn:ou=people,dc=example,dc=com \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

   You can optionally specify unique values across multiple base DNs:

```
$ dsconfig \
 set-plugin-prop \
 --hostname localhost \
 --port 4444 \
 --bindDn uid=admin \
 --bindPassword password \
 --plugin-name "UID Unique Attribute" \
 --set enabled:true \
 --add base-dn:ou=people,dc=example,dc=com \
 --add base-dn:ou=people,dc=example,dc=org \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
```

```
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

2. Check your work:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: uid=ajensen,ou=People,dc=example,dc=com
changetype: modify
add: uid
uid: bjensen
EOF

# The LDAP modify request failed: 19 (Constraint
Violation)
# Additional Information:  A unique attribute conflict was
detected for attribute uid: value bjensen already exists
in entry uid=bjensen,ou=People,dc=example,dc=com
```

If you have set up multiple base DNs, check your work as follows:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: uid=bjensen,ou=People,dc=example,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Babs
sn: Jensen
uid: bjensen
```

```
EOF

# The LDAP modify request failed: 19 (Constraint
Violation)
# Additional Information:  A unique attribute conflict was
detected for attribute uid: value bjensen already exists
in entry uid=bjensen,ou=People,dc=example,dc=com
```

Make Other Attributes Unique

You can configure the unique attribute plugin for use with any attributes, not just `uid`:

1. Before you set up the plugin, index the attribute for equality.

   For instructions, see Configure Indexes.

2. Set up the plugin configuration for your attribute using one of the following alternatives:

   a. Add the attribute to an existing plugin configuration:

   ```
   $ dsconfig \
    set-plugin-prop \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --plugin-name "UID Unique Attribute" \
    --add type:telephoneNumber \
    --usePkcs12TrustStore /path/to/opendj/config/keystore
   \
    --trustStorePassword:file
   /path/to/opendj/config/keystore.pin \
    --no-prompt
   ```

   Choose this alternative if you want each value to be unique across all configured attributes.

   b. Create a new plugin configuration:

   ```
   $ dsconfig \
    create-plugin \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
   ```

```
 --bindPassword password \
 --plugin-name "Unique phone numbers" \
 --type unique-attribute \
 --set enabled:true \
 --set base-dn:ou=people,dc=example,dc=com \
 --set type:telephoneNumber \
 --usePkcs12TrustStore /path/to/opendj/config/keystore
\
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

Choose this alternative if values only need to be unique within the context of a particular attribute.

3. Check your work:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: uid=ajensen,ou=People,dc=example,dc=com
changetype: modify
replace: telephoneNumber
telephoneNumber: +1 828 555 1212

dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: telephoneNumber
telephoneNumber: +1 828 555 1212
EOF

# MODIFY operation successful for DN
uid=ajensen,ou=People,dc=example,dc=com

# The LDAP modify request failed: 19 (Constraint
Violation)
# Additional Information:  A unique attribute conflict was
detected for attribute telephoneNumber: value +1 828 555
```

```
1212 already exists in entry
uid=ajensen,ou=People,dc=example,dc=com
```

Scope Uniqueness

In some cases, attributes must be unique, but only in the context of a particular base DN. For example, `uid` values must be unique under `dc=example,dc=com` and under `dc=example,dc=org`. But it is okay to have `uid=bjensen,ou=people,dc=example,dc=com` and `uid=bjensen,ou=people,dc=example,dc=org`:

1. If the attribute you target is not indexed for equality by default, index the attribute for equality.

   See Configure Indexes for instructions.

2. For each base DN, set up a configuration entry that ensures the target attribute values are unique:

```
$ dsconfig \
 create-plugin \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --plugin-name "Unique Example.com UIDs" \
 --type unique-attribute \
 --set enabled:true \
 --set base-dn:dc=example,dc=com \
 --set type:uid \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
$ dsconfig \
 create-plugin \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --plugin-name "Unique Example.org UIDs" \
 --type unique-attribute \
 --set enabled:true \
 --set base-dn:dc=example,dc=org \
```

```
 --set type:uid \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. Check your work:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: uid=unique,ou=People,dc=example,dc=com
uid: unique
givenName: Unique
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: Unique Person
sn: Person
userPassword: 1Mun1qu3

dn: uid=unique,ou=People,dc=example,dc=org
uid: unique
givenName: Unique
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: Unique Person
sn: Person
userPassword: 1Mun1qu3

dn: uid=copycat,ou=People,dc=example,dc=com
uid: unique
uid: copycat
givenName: Copycat
objectClass: person
objectClass: organizationalPerson
```

```
objectClass: inetOrgPerson
objectClass: top
cn: Copycat Person
sn: Person
userPassword: copycopy
EOF

# ADD operation successful for DN
uid=unique,ou=People,dc=example,dc=com

# The LDAP modify request failed: 19 (Constraint
Violation)
# Additional Information:  A unique attribute conflict was
detected for attribute uid: value unique already exists in
entry uid=unique,ou=People,dc=example,dc=com
```

Use Uniqueness With Replication

The unique attribute plugin only ensures uniqueness on the replica where the attribute is updated. If client applications write the same attribute value separately at the same time on different replicas, both replicas might use the same "unique" value, especially if the network is down between the replicas:

1. Configure the plugin identically on all replicas.
2. To avoid duplicate values where possible, use DS directory proxy to direct all updates of the unique attribute to the same replica.

Samba Password Sync

Samba, the Windows interoperability suite for Linux and UNIX, stores accounts because UNIX and Windows password storage management is not interoperable. The default account storage mechanism works well with small numbers of accounts and one domain controller. For larger installations, Samba can use DS replicas to store Samba accounts. See the Samba documentation for your platform for instructions on how to configure LDAP directory servers as Samba `passdb` backends.

The procedures that follow focus on how to keep passwords in sync for Samba account storage.

When you store Samba accounts in a directory server, Samba stores its own attributes as defined in the Samba schema. Samba does not use the LDAP standard

`userPassword` attribute to store users' Samba passwords. You can configure Samba to apply changes to Samba passwords to LDAP passwords as well. Yet, if a user modifies their LDAP password directly without updating the Samba password, the LDAP and Samba passwords get out of sync.

The DS Samba Password plugin resolves this problem for you. The plugin intercepts password changes to Samba user profiles, synchronizing Samba password and LDAP password values. For an incoming Password Modify Extended Request or modify request to change the user password, the DS Samba Password plugin detects whether the user's entry is a Samba user profile (entry has object class `sambaSAMAccount`), hashes the incoming password value, and applies the password change to the appropriate password attribute, keeping the password values in sync. The DS Samba Password plugin can perform synchronization as long as new passwords are provided in plaintext in the modification request. If you configure Samba to synchronize LDAP passwords when it changes Samba passwords, the plugin can ignore changes by the Samba user to avoid duplicate synchronization.

Create the Samba Administrator

The Samba Administrator updates the LDAP password when a Samba password changes.

In Samba's `smb.conf` configuration file, the value of `ldap admin dn` is set to the DN of this account. When the Samba Administrator changes a user password, the plugin ignores the changes. Choose a distinct account different from the directory superuser and other administrators:

1. Create or choose an account for the Samba Administrator:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: uid=Samba Admin,ou=Special Users,dc=example,dc=com
cn: Samba Administrator
givenName: Samba
mail: samba@example.com
objectClass: person
objectClass: inetOrgPerson
```

```
objectClass: organizationalPerson
objectClass: top
sn: Administrator
uid: Samba Admin
userPassword: chngthspwd
EOF
```

2. Let the Samba Administrator reset user passwords:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: uid=Samba Admin,ou=Special Users,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: password-reset

dn: dc=example,dc=com
changetype: modify
add: aci
aci: (target="ldap:///dc=example,dc=com")(targetattr ="*")
 (version 3.0; acl "Samba Admin user rights"; allow(all)
  userdn="ldap:///uid=Samba Admin,ou=Special
Users,dc=example,dc=com";)
EOF
```

Enable the Samba Password Plugin

1. Determine whether the plugin must store passwords hashed like LanManager (`sync-lm-password`) or like Windows NT (`sync-nt-password`), based on the Samba configuration.
2. Enable the plugin:

```
$ dsconfig \
 create-plugin \
 --hostname localhost \
```

```
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --plugin-name "Samba Password Synchronisation" \
 --type samba-password \
 --set enabled:true \
 --set pwd-sync-policy:sync-nt-password \
 --set samba-administrator-dn:"uid=Samba Admin,ou=Special
Users,dc=example,dc=com" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

The Samba Password plugin is active immediately.

3. When troubleshooting Samba Password plugin issues, turn on debug logging:

```
$ dsconfig \
 create-debug-target \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --target-name
org.opends.server.plugins.SambaPasswordPlugin \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 set-log-publisher-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

```
$ tail -f /path/to/opendj/logs/debug
```

LDAP Proxy

Directory Services proxy services let you build a single point of access to a directory service, with a uniform view of the underlying LDAPv3 user data. This hides implementation details from directory client applications, and promotes scalability and availability.

When you set up a directory proxy server, no user data is stored locally. The server acts purely as an LDAP proxy. The only local data is for the server configuration and LDAP schema. In addition, the server is set up to use global access control policies rather than global ACIs. Global access control policies provide coarse-grained access control suitable for use on proxy servers, where the lack of local access to directory data makes ACIs a poor fit.

Proxy services are provided by proxy backends. Proxy backends connect to remote directory servers using a dynamic and configurable discovery mechanism. They route requests to remote directory servers. The way they distribute requests is configurable. The way they handle failures when processing forwarded requests.

LDAP schema definitions for user data must be aligned on the proxy server and on the remote directory servers.

ACIs are handled by the directory server where the target data is stored. In other words, global access control policies set on a proxy server do not change ACIs on the remote directory server. Set ACIs appropriately on the directory server independently of proxy settings.

Remote LDAP Servers

When the target DN of an LDAP request is not in a local backend, an LDAP server can refuse to handle the request, or return a referral. An LDAP proxy can also forward the request to another directory server.

In Directory Services, the LDAP proxy is implemented as a proxy backend. Rather than store user data locally, the proxy backend forwards requests to remote directory servers.

For proxy backends, a *service discovery mechanism* identifies remote directory servers to forward LDAP requests to. A service discovery mechanism's configuration specifies the keys used for secure communications, and how to contact the remote directory servers. It reads remote directory servers' configurations to discover their capabilities, including

the naming contexts they serve, and so which target DNs they can handle. It periodically rereads their configurations in case they have been updated since the last service discovery operation.

When preparing to configure a service discovery mechanism, choose one of these alternatives:

Replication service discovery mechanism
: This mechanism contacts DS replication servers to discover remote LDAP servers. Each replication server maintains information about the replication topology that lets the proxy server discover directory server replicas.

  This mechanism only works with replicated DS servers.

  A replication service discovery mechanism configuration includes a bind DN and password to connect to replication servers. It uses this account to read configuration data. The account must have access and privileges to read the configuration data, and it must exist with the same credentials on all replication servers.

Static service discovery mechanism
: This mechanism maintains a static list of directory server `host:port` combinations. You must enumerate the remote LDAP servers.

  This mechanism is designed to work with all LDAPv3 directory servers that support proxied authorization.

When configuring a service discovery mechanism, make sure all the remote directory servers are replicas of each other, and that they have the same capabilities. A proxy backend expects all remote directory server replicas known to the mechanism to hold the same data. This allows the backend to treat the replicas as equivalent members of a pool. In the configuration, a pool of equivalent replicas is a *shard*.

In deployments where you must distribute data for horizontal write scalability, you can configure multiple service discovery mechanisms. The proxy backend can then distribute write requests across multiple shards.

The following example creates a replication service discovery mechanism that specifies two replication servers:

```
$ dsconfig \
 create-service-discovery-mechanism \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --mechanism-name "Replication Service Discovery Mechanism" \
```

```
 --type replication \
 --set bootstrap-replication-server:rs1.example.com:4444 \
 --set bootstrap-replication-server:rs2.example.com:4444 \
 --set ssl-cert-nickname:ssl-key-pair \
 --set key-manager-provider:PKCS12 \
 --set trust-manager-provider:PKCS12 \
 --set use-start-tls:true \
 --set use-sasl-external:true \
 --no-prompt
```

The example above assumes that the servers protect connections using keys generated with a deployment key and password. If this is not the case, configure the security settings appropriately.

The following example creates a static service discovery mechanism that specifies four remote LDAP servers:

```
$ dsconfig \
 create-service-discovery-mechanism \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --mechanism-name "Static Service Discovery Mechanism" \
 --type static \
 --set primary-server:local1.example.com:636 \
 --set primary-server:local2.example.com:636 \
 --set secondary-server:remote1.example.com:636 \
 --set secondary-server:remote2.example.com:636 \
 --set ssl-cert-nickname:ssl-key-pair \
 --set key-manager-provider:PKCS12 \
 --set trust-manager-provider:"JVM Trust Manager" \
 --set use-ssl:true \
 --set use-sasl-external:true \
 --no-prompt
```

The example above assumes that the remote servers use certificates signed by well-known CAs, and so are recognized using the trust manager provided by the JVM. If this is not the case, configure the security settings appropriately.

If the proxy server must perform SSL mutual authentication when setting up secure connections with the remote servers, configure an appropriate key manager provider and SSL certificate nickname.

Supported service discovery mechanisms define a `discovery-interval` that specifies how often to read the remote server configurations to discover changes. Because the mechanism polls periodically for configuration changes, by default, it can take up to one minute for the mechanism to see the changes. If necessary, you can change this setting in the configuration.

Routing Requests

A proxy backend forwards requests according to their target DNs. The proxy backend matches target DNs to base DNs that you specify in the configuration. If you specify multiple shards for data distribution, the proxy also forwards requests to the appropriate shard.

When specifying base DNs, bear in mind the following points:

- A server responds first with local data, such as the server's own configuration or monitoring data. If a request target DN cannot be served from local data, the proxy backend can forward the request to a remote directory server.

  The proxy backend will forward the request if its target DN is under one of the specified base DNs.

- As an alternative to enumerating a list of base DNs to proxy, you can set the proxy backend property `route-all: true`.

  When you activate this property, the proxy backend will attempt to forward all requests that can be served by public naming contexts of remote servers. If the request target DN is not in a naming context supported by any remote directory servers associated with the proxy backend, then the proxy will not forward the request.

- The LDAP proxy capability is intended to proxy user data, not server-specific data.

  A server with a proxy backend still responds directly to requests that target private naming contexts, such as `cn=config`, `cn=tasks`, and `cn=monitor`. Local backends hold configuration and monitoring information that is specific to the server, and these naming contexts are not public.

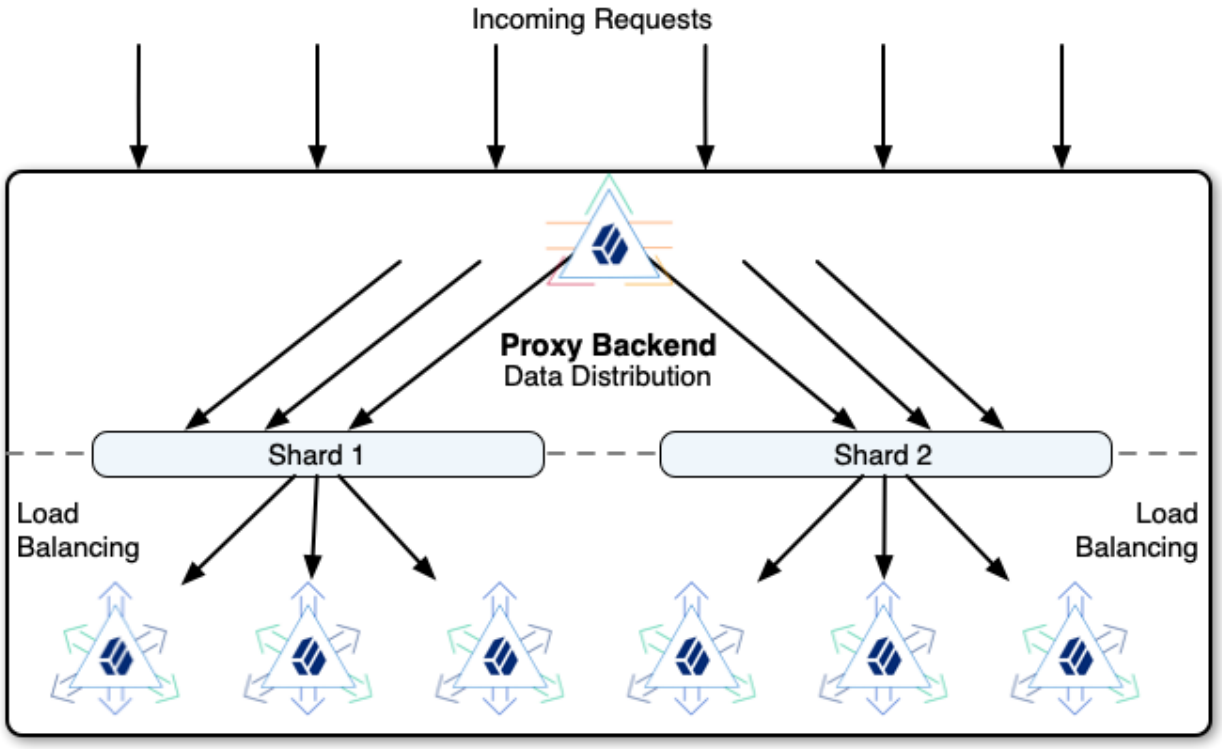
  Make sure client applications for configuration and monitoring access servers directly for the server-specific data they need.

- If you configure multiple proxy backends, each proxy backend must target distinct base DNs.

In deployments where you must distribute data for horizontal write scalability, you can configure multiple service discovery mechanisms for the same base DNs. Each service discovery mechanism targets its own distinct shard of directory data.

To enable write scalability beyond what is possible with a single shard of replicas, each shard must have its own independent replication configuration. Replication replays each update only within the shard.

The proxy backend distributes write requests across the shard, and balances the load within each shard:



Data distribution for horizontal scalability has the following properties:

- The proxy backend always routes requests targeting an entry below the partition base DN to the *same* shard.
- The proxy backend routes read requests targeting the partition base DN entry or above to *any* shard.

  In other words, the proxy backend can route each request to a different shard. When you deploy data distribution, the proxy rejects writes to the partition base DN entry and above through the proxy backend. Instead, you must perform such write operations on a replica in each shard.

  The best practice is therefore to update the partition base DN entry and above prior to deployment.
- The proxy backend routes search requests to *all* shards unless the search scope is below the partition base DN.

  For example, if the partition base DN is `ou=people,dc=example,dc=com`, the proxy backend always routes a search with base DN `uid=bjensen,ou=people,dc=example,dc=com` or `deviceid=12345,uid=bjensen,ou=people,dc=example,dc=com` to the same

shard. However, it routes a search with base DN `ou=people,dc=example,dc=com` to all shards.

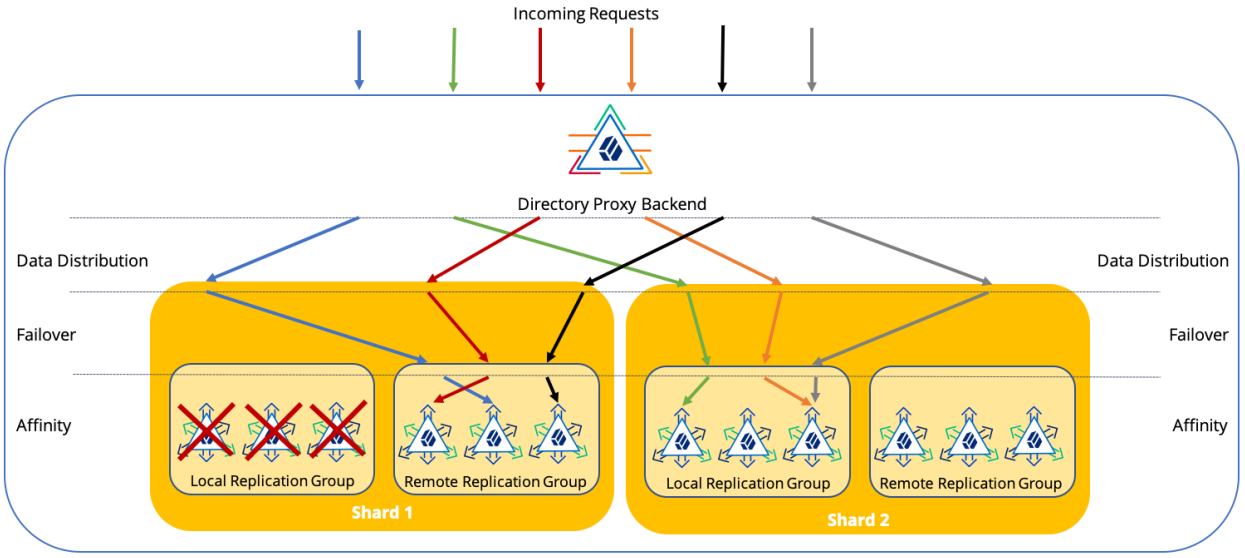
Load Balancing

A directory proxy server balances the LDAP request load across the remote LDAP servers.

The proxy performs load balancing for the following purposes:

- **Data distribution**: sharding data to scale out
- **Failover**: routing requests to available directory servers
- **Affinity**: consistently forwarding requests for the same entry to the same server

The proxy applies these load balancing capabilities in hierarchical order to route requests. Follow the paths of requests through the proxy backend to a directory server in this diagram:



Notice in the diagram how the load balancing capabilities work together to route a request to a directory server:

- Data distribution routes to the correct shard.
- Failover routes around directory servers that are down, routing the request to available servers.
- Affinity routes the operation to a specific server.

Feature	Characteristics
Data Distribution	Routes requests with the same target DN below the partition base DN to the same shard. Data distribution helps to scale out the directory service horizontally. *When to Use* Use when you must scale out the directory service, and the deployment fits the constraints. A single DS directory service shard can process thousands of LDAP requests *per second* given the proper configuration and tuning. Furthermore, you can increase the search and read throughput simply by adding replicas. Configure data distribution for deployments when performance testing demonstrates that you cannot achieve write throughput requirements with properly configured and tuned replicas in a single replication topology. For example, use data distribution for a very high scale AM CTS deployment. *Settings* See Data Distribution.

Feature	Characteristics
Failover	Routes LDAP requests to LDAP servers that are available and responding to health check probe requests. Failover prevents the proxy server from continuing to forward requests to unavailable servers. See also About Failures. *When to Use* Use failover settings to route requests to local servers if they are available, and remote servers only when local servers are not available. *Settings* Proxy backend properties: • `connection-timeout` • `heartbeat-interval` • `heartbeat-search-request-base-dn` Replication service discovery mechanism property: • `primary-group-id` Static service discovery mechanism properties: • `primary-server` • `secondary-server`

Feature	Characteristics
Affinity	Routes LDAP requests with the same target DN to the same server. Affinity load balancing helps applications that update and then reread the same entry in quick succession. This is not a best practice, but is often seen in client applications. With an add or modify request on an entry that is quickly followed by a read of the entry, the requests to replicate the update can take longer than the read request, depending on network latency. Affinity load balancing forwards the read request to the same server that processed the update, ensuring that the client application obtains the expected result. Affinity routing depends on the values of the proxy backend property, `partition-base-dn`. The proxy consistently routes requests for entries subordinate to these entries to the same server. The values of this property should therefore be the lowest entries in your DIT that are part of the DIT structure and not part of application data. In other words, when using affinity with two main branches, `ou=groups,dc=example,dc=com` and `ou=people,dc=example,dc=com`, set: • `partition-base-dn:ou=groups,dc=example,dc=com` • `partition-base-dn:ou=people,dc=example,dc=com` In terms of the CAP theorem, affinity load balancing provides consistency and availability, but not partition tolerance. As this algorithm lacks partition tolerance, configure it to load balance requests in environments where partitions are unlikely, such as a single data center with all directory servers on the same network. *When to Use* Use whenever possible, in combination with failover. Client application developers may be unaware of LDAP best practices. *Settings* Proxy backend properties: • `partition-base-dn`

LDAP Schema

Proxy services are designed to proxy user data rather than server configuration or monitoring data. A proxy server must expose the same LDAP schema for user data as the remote directory servers.

If the schema definitions for user data differ between the proxy server and the remote directory servers, you must update them to bring them into alignment.

For details on DS server schema, see LDAP Schema.

If the remote servers are not DS servers, see the schema documentation for the remote servers. Schema formats are not identical across all LDAPv3 servers. You will need to translate the differences into native formats, and apply changes locally on each server.

Proxy Backend

Create proxy backends only on servers set up as proxy servers (with **`setup --profile ds-proxy-server`**).

IMPORTANT

IMPORTANT

A directory proxy server connects using an account that must exist in the remote directory service.

The directory proxy server binds with this service account, and then forwards LDAP requests on behalf of other users.

For DS directory servers, use the proxied server setup profile if possible. For details, see Install DS For Use With DS Proxy.

The service account must have the following on all remote directory servers:

- The same bind credentials.

  If possible, use mutual TLS to authenticate the proxy user with the backend servers.

- The right to perform proxied authorization.

  Make sure the LDAP servers support proxied authorization (control OID: `2.16.840.1.113730.3.4.18`).

  For details, see RFC 4370, *Lightweight Directory Access Protocol (LDAP) Proxied Authorization Control*.

- When using a replication discovery mechanism with remote DS directory servers, the service account requires the `config-read` and `monitor-read` privileges for the service discovery mechanism. It requires the `proxied-auth` privilege and an ACI to perform proxied authorization.

The following listing shows an example service account that you could use with DS replicas. Adapt the account as necessary for your directory service:

```
dn: uid=proxy
objectClass: top
objectClass: account
objectClass: ds-certificate-user
uid: proxy
ds-certificate-subject-dn: CN=DS, O=ForgeRock.com
ds-privilege-name: config-read
ds-privilege-name: monitor-read
ds-privilege-name: proxied-auth
aci: (targetcontrol="ProxiedAuth")
  (version 3.0; acl "Allow proxied authorization";
  allow(read) userdn="ldap:///uid=proxy";)
```

Forward for Example.com

The following example creates a proxy backend that forwards LDAP requests when the target DN is in `dc=example,dc=com`:

```
$ dsconfig \
 create-backend \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --backend-name proxyExampleCom \
 --type proxy \
 --set enabled:true \
 --set base-dn:dc=example,dc=com \
 --set route-all:false \
 --set use-sasl-external:true \
 --set ssl-cert-nickname:ssl-key-pair \
 --set key-manager-provider:PKCS12 \
 --set partition-base-dn:ou=groups,dc=example,dc=com \
 --set partition-base-dn:ou=people,dc=example,dc=com \
 --set shard:"Replication Service Discovery Mechanism" \
 --set heartbeat-search-request-base-
dn:cn=proxy,ou=apps,dc=example,dc=com \
 --no-prompt
```

This command fails if `dc=example,dc=com` is already served by local data.

Notice the setting for `heartbeat-search-request-base-dn`. By default, the proxy backend sends periodic heartbeat requests to remote directory servers to help manage its connections. About Failures describes in more detail how a proxy backend uses heartbeats.

By default, heartbeat requests target the remote directory server root DSE. A response from the root DSE suffices to determine whether the server is up, but it does not indicate whether the proxy can access data under a given base DN. Set `heartbeat-search-request-base-dn` to target an entry located under a base DN of interest.

Forward All Requests

The following example creates a proxy backend that forwards LDAP requests targeting user data. It does not specify base DNs explicitly, but uses the `route-all` property:

```
$ dsconfig \
 create-backend \
```

```
--hostname localhost \
--port 4444 \
--bindDN uid=admin \
--bindPassword password \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--backend-name proxyAll \
--type proxy \
--set enabled:true \
--set route-all:true \
--set use-sasl-external:true \
--set ssl-cert-nickname:ssl-key-pair \
--set key-manager-provider:PKCS12 \
--set shard:"Static Service Discovery Mechanism" \
--no-prompt
```

The example above assumes that the servers protect connections using keys generated with a deployment key and password. If this is not the case, configure the security settings appropriately.

Proxy backends define a `discovery-interval` that specifies how often to read the remote server configurations to discover changes. Because the proxy polls periodically for configuration changes, by default, it can take up to one minute to see the changes. If necessary, you can change this setting in the configuration.

About Failures

There are many specific ways that an LDAP request can fail. Not all failure result codes indicate a permanent failure, however. The following result codes from a remote directory server indicate a temporary failure:

- 51 (Busy) indicates that the server was too busy to process the request.

  The request can safely be tried on another peer server.

- 52 (Unavailable) indicates that the server was missing at least one resource needed to process the request.

  The request can safely be tried on another peer server.

When a forwarded request finishes with one of these return codes, the proxy backend retries the request on another server. In this case, the client does not receive the result from the remote directory server.

When a forwarded request finishes with a permanent server-side error return code, the proxy backend returns the result to the client application. In this case, the client must handle the error.

Connection failures can prevent the remote directory server from responding at all. The proxy backend handles connection failures differently, depending on whether it is inherently safe to replay the operation:

- For operations that read directory data, including search and compare requests, the proxy backend retries the request if a connection failure occurs.
- For operations that write directory data, including add, delete, modify, and modify DN requests, the proxy backend does not retry the request if a connection failure occurs.

  When the connection fails during a write operation, it is not possible to determine whether the change was applied. It is not safe, therefore, to replay the request on another server.

  The proxy returns an error to the client application.
- Connection failures during bind operations cause the proxy to retry the bind.

  In the unlucky event of repeated connection failures on successive bind attempts combined with a password policy configured to lock an account after a certain number of consecutive failures, it is possible that this behavior could lock an account.

A proxy backend protects against failed and stale connections with feedback from periodic heartbeat requests to remote directory servers.

A heartbeat request serves these purposes:

- Checks that the remote directory server is still available.

  If the request fails, the proxy closes the unresponsive connection and connects to another remote directory server.
- Determines whether an unresponsive server has recovered.

  When the remote directory server responds again to heartbeat requests, the proxy can begin forwarding client requests to it again.
- Keeps the connection alive, preventing it from appearing idle and being forcefully closed.

If necessary, you can configure how often heartbeat requests are sent using the proxy backend `heartbeat-interval` property.

Access Control and Resource Limits

When you deploy a directory proxy server, you limit the mechanisms for access control and resource limits. Such mechanisms cannot rely on ACIs in user data, resource limit settings on user entries, or group membership to determine what the server should

allow. They can depend only on the server configuration and on the properties of the incoming request.

Global access control policies provide a mechanism suitable for directory proxy servers. For details, see Access Control.

Resource limits set in the server configuration provide a way to prevent directory clients from using an unfair share of system resources. For details, see Resource Limits, which covers how to update a server's configuration.

High Availability

Directory services are designed for basic availability. Directory data replication makes it possible to read and write directory data when the network is partitioned, where remote servers cannot effectively contact each other. This sort of availability assumes that client applications can handle temporary interruptions.

Some client applications can be configured to connect to a set of directory servers. Some of these clients are able to retry requests when a server is unavailable. Others expect a single point of entry to the directory service, or cannot continue promptly when a directory server is unavailable.

Individual directory servers can become unavailable for various reasons:

- A network problem can make it impossible to reach the server.
- The server can be temporarily down for maintenance (backup, upgrade, and other purposes).
- The server can be removed from a replication topology and replaced with a different server.
- A crash can bring the server down temporarily for a restart or permanently for a replacement.

All of these interruptions must be managed on the client side. Some could require reconfiguration of each client application.

A directory proxy server provides high availability to client applications by hiding these implementation details from client applications. The proxy presents client applications with a uniform view of the remote directory servers. It periodically rereads the remote servers' configurations and checks connections to route requests correctly despite changes in server configurations and availability.

Directory proxy servers are well-suited for applications that need a single entry point to the directory service.

Single Point of Access

Unlike directory servers with their potentially large sets of volatile user data, directory proxy servers manage only their configuration state. Proxy servers can start faster and require less disk and memory space. Each directory proxy server can effectively be a clone of every other, with a configuration that does not change after server startup. Each clone routes LDAP requests and handles problems in the same way.

When you configure all directory proxy servers as clones of each other, you have a number of identical directory access points. Each point provides the same view of the underlying directory service. The points differ from each other only by their connection coordinates (host, port, and potentially key material to establish secure connections).

To consolidate identical access points to a single access point, configure your network to use a virtual IP address for the proxy servers. You can restart or replace proxy servers at any time, in the worst case losing only the client connections that were established with the individual proxy server.

An alternative to a single, global access point for all applications is to repeat the same approach for each key application. The proxy server configuration is specific to each key application. Clones with that configuration provide a single directory access point for the key application. Other clones do the same other for other key applications. Be sure to provide a more generic access point for additional applications.

Data Distribution

Data distribution through the proxy comes with the following constraints:

- Data distribution is not elastic, and does not redistribute data if you add a shard. Once you deploy and use data distribution, you cannot change the number of shards.

  Plan for peak load when using data distribution to scale out your deployment.

  You can, however, add or change individual servers within a shard. For example, you could scale out by deploying many shards, and later upgrade to more powerful, faster underlying systems to scale up each shard.
- You cannot import distributed data from LDIF. The **`import-ldif`** command does not work with a proxy backend.

  If you must initialize distributed data, add the entries through the proxy server instead. You can use the **`ldapmodify`** command, for example, setting the `--numConnections` option to perform updates in parallel on multiple LDAP connections. You can also deploy as many proxy servers as necessary to perform the adds in parallel.
- Write requests for entries above the distributed data are not repeated on all shards. Instead, the proxy rejects such requests.

If you must make a change to entries above the distributed data, make the change behind the proxy to one directory server replica in each shard.

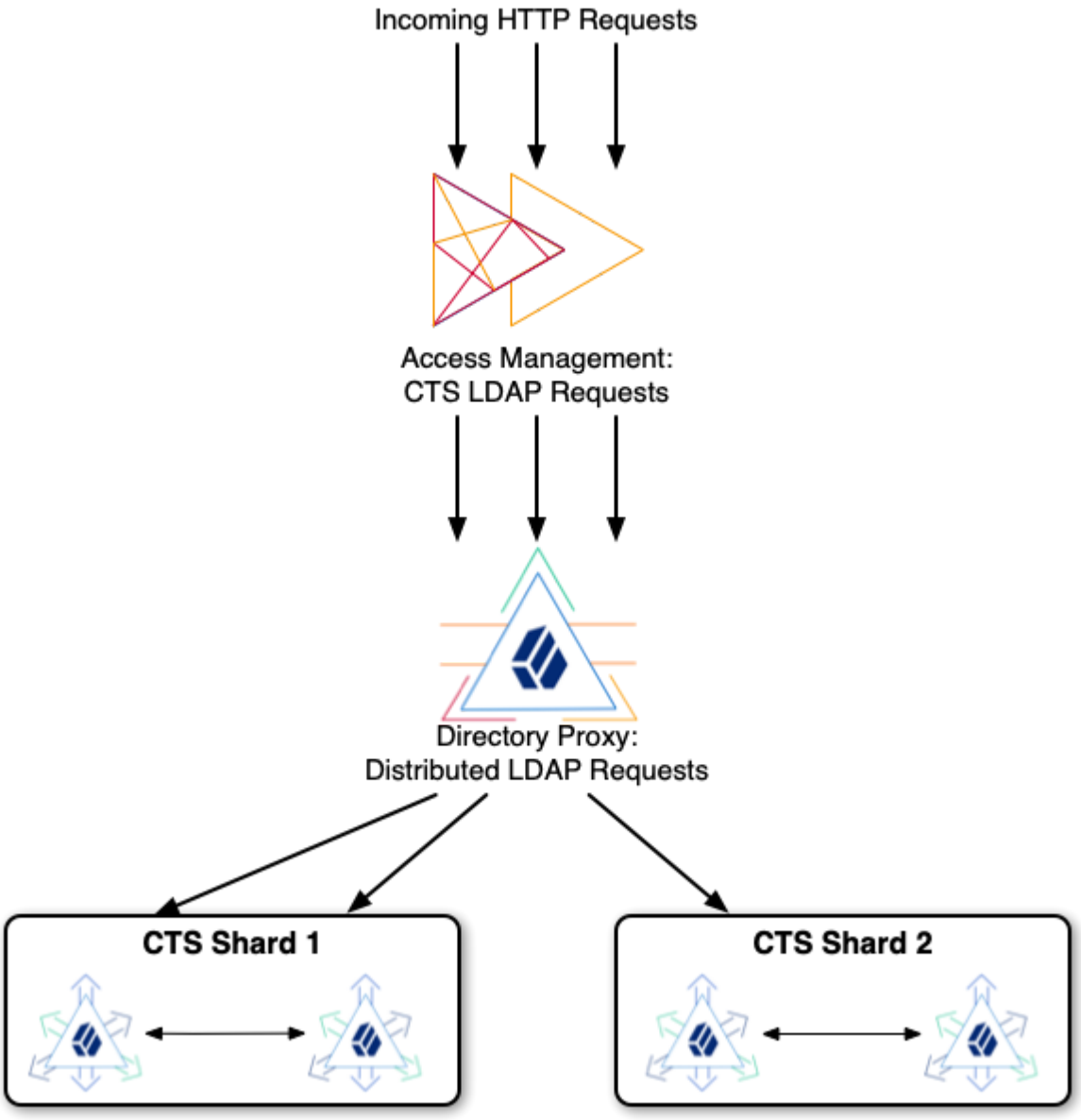
- Subtree and one-level searches can be relatively expensive, as the proxy must potentially forward the search request to every shard to retrieve all search results.

  To optimize searches, use a search base DN that points to a distributed entry.

- Given the present constraints of both data distribution and replication, the best fit for data distribution occurs where the distributed data remains self-contained and logically independent from other data branches.

  The example shown below distributes data for AM CTS tokens. The CTS data model fits DS data distribution well. AM stores all CTS token entries under the same DN. The entries above the distributed CTS tokens do not change during normal operation. CTS token entries do not have DN-value attributes that reference entries in another branch, nor are they members of LDAP groups stored in another branch.

The example that follows is intended for evaluation on a single computer. It involves installing five DS servers and one AM server:



As you follow the example, notice the following characteristics:

- The example is intended for evaluation only.

  In production, deploy each server on a separate system, and use secure connections.
- The AM server connects to the DS proxy for CTS LDAP requests.
- The DS proxy distributes LDAP requests across two shards.
- Each shard holds two DS replicas set up with the AM CTS profile.

  DS servers replicate only *within* shards. Servers in different partitions never replicate to each other. If replication crosses a partition boundary, the data is replicated everywhere. Replicating data everywhere would defeat the purpose of data distribution.

Try the CTS Example

1. Make sure you have the Bash shell installed so that you can run the scripts to install and configure DS servers.
2. Download the DS .zip delivery and keep track of where you saved it.

   The scripts use the file name, `~/Downloads/DS-7.1.8.zip`.
3. Stop services that use the following ports. Servers in the example cannot start if these ports are in use:
   - `1636`
   - `4444`
   - `5444`
   - `5636`
   - `5989`
   - `6444`
   - `6636`
   - `6989`
   - `8080`
   - `15444`
   - `15636`
   - `15989`
   - `16444`
   - `16636`

- 16989

4. Set up the DS replicas:
   - Save a local copy of the script: setup-cts-replicas.sh.

     ▼ *View script*

```
#!/usr/bin/env bash
#
# Copyright 2018-2020 ForgeRock AS. All Rights
Reserved
#
# Use of this code requires a commercial software
license with ForgeRock AS.
# or with one of its affiliates. All use shall be
exclusively subject
# to such license between the licensee and ForgeRock
AS.
#

###
# Set up directory server replicas for CTS in
appropriate shards.
# This is intended for evaluation on a single system.
#
# In deployment, each of these replicas would be on a
separate host system.
#
# The proxy distributes data across two CTS shards,
each with two replicas.
#
# Each shard is served by a pool of separate
replicas.
# In other words, each server replicates within one
shard only.
#
# All servers have the same uid=Proxy service
account.
# The proxy binds with uid=Proxy and uses proxied
authorization.
# This script adds an ACI to allow the proxy service
account
# to perform proxied authorization under the CTS
shards.
#
# All CTS shards have a CTS admin account,
```

```
uid=openam_cts,ou=tokens.
# To modify the account, for example to change the
password,
# you must modify it once for each shard, not through
the proxy.
# The proxy would distribute the change to only one
shard.
#
# The shards have the following replication
configurations,
# with each server's (admin-port ldaps-port):
# cts 1: (5444 5636) <==> (15444 15636)
# cts 2: (6444 6636) <==> (16444 16636)
###

###
# Adjust these variables to fit your circumstances:
ZIP=~/Downloads/DS-7.1.8.zip
BASE_DIR=/path/to
FQDN=localhost
DEPLOYMENT_KEY=AMdhPkZJxwoEjwx3zV1IGOccAHNvvQ5CBVN1bk
VDdmCLA99ueRm6Cg
DEPLOYMENT_PASSWORD=password
###

CURRENT_DIR=$(pwd)

# Install a single DS/RS with the CTS profile from
the .zip distribution.
# The AM CTS reaper will manage token expiration and
deletion.
# $1: instance number
# $2: bootstrap server base
install() {
  echo "### Installing ${BASE_DIR}/ds-rs-${1} ###"
  unzip -q "${ZIP}"
  mv opendj "ds-rs-${1}"

  "${BASE_DIR}/ds-rs-${1}/setup" \
   --deploymentKey "$DEPLOYMENT_KEY" \
   --deploymentKeyPassword "$DEPLOYMENT_PASSWORD" \
   --serverId "ds-rs-${1}" \
   --adminConnectorPort "${1}444" \
   --hostname "${FQDN}" \
   --ldapsPort "${1}636" \
```

```
    --enableStartTls \
    --replicationPort "${1}989" \
    --bootstrapReplicationServer "${FQDN}:${2}989" \
    --bootstrapReplicationServer "${FQDN}:1${2}989" \
    --rootUserDN uid=admin \
    --rootUserPassword password \
    --profile am-cts \
    --set am-cts/amCtsAdminPassword:password \
    --profile ds-proxied-server \
    --set ds-proxied-server/baseDn:ou=tokens \
    --acceptLicense

  echo "### Starting ds-rs-${1} ###"
  "${BASE_DIR}/ds-rs-${1}/bin/start-ds" --quiet
}

move_cts_admin() {
  echo "### Moving CTS admin account above the
distributed data ###"
  "${BASE_DIR}/ds-rs-${1}/bin/ldapmodify" \
    --hostname "${FQDN}" \
    --port "${1}636" \
    --useSsl \
    --usePkcs12TrustStore "${BASE_DIR}/ds-
rs-${1}/config/keystore" \
    --trustStorePassword:file "${BASE_DIR}/ds-
rs-${1}/config/keystore.pin" \
    --bindDn uid=admin \
    --bindPassword password << EOF
dn: uid=openam_cts,ou=admins,ou=famrecords,ou=openam-
session,ou=tokens
changetype: moddn
newrdn: uid=openam_cts
deleteoldrdn: 0
newsuperior: ou=tokens
EOF
}

add_aci() {
  echo "### Adding ACIs for moved CTS admin account
on ds-rs-${1} ###"
  "${BASE_DIR}/ds-rs-${1}/bin/ldapmodify" \
    --hostname "${FQDN}" \
    --port "${1}636" \
    --useSsl \
```

```
   --usePkcs12TrustStore "${BASE_DIR}/ds-
rs-${1}/config/keystore" \
   --trustStorePassword:file "${BASE_DIR}/ds-
rs-${1}/config/keystore.pin" \
   --bindDn uid=admin \
   --bindPassword password <<EOF
dn: ou=tokens
changetype: modify
add: aci
aci: (targetattr="*") (version 3.0; acl "Allow read
access for debugging";
  allow(read, search, compare) userdn =
"ldap:///uid=openam_cts,ou=tokens";)

dn: ou=famrecords,ou=openam-session,ou=tokens
changetype: modify
add: aci
aci: (targetattr="*") (version 3.0; acl "Create,
delete, update token entries";
  allow(add, delete, write) userdn =
"ldap:///uid=openam_cts,ou=tokens";)
EOF
}

cd "${BASE_DIR}" || exit 1

for i in 5 6
do
  install ${i} ${i}
  install 1${i} ${i}
done

for i in 5 6
do
  move_cts_admin ${i}
  add_aci ${i}
done

# In this example, all replicas have the same data to
start with.
# If the data is not identical on each pair of
replicas, initialize replication manually.

cd "${CURRENT_DIR}" || exit
```

- If necessary, edit the script for use on your computer.
- Run the script.

  After the script finishes successfully, four DS replicas in two separate partitions are running on your computer.

5. Set up the DS proxy:
   - Save a local copy of the script: setup-cts-proxy.sh.

     ▼ *View script*

```
#!/usr/bin/env bash
#
# Copyright 2018-2020 ForgeRock AS. All Rights
Reserved
#
# Use of this code requires a commercial software
license with ForgeRock AS.
# or with one of its affiliates. All use shall be
exclusively subject
# to such license between the licensee and ForgeRock
AS.
#

###
# Set up a directory proxy server listening on the
typical ports.
# This is intended for evaluation on a single system.
#
# In deployment, this would be a layer of identical
proxy servers
# to balance incoming requests.
#
# This server distributes data across two replication
shards for CTS.
# Each shard is served by two replicas.
# Each DS directory server replicates within one
shard only.
###

###
# Adjust these variables to fit your circumstances:
ZIP=~/Downloads/DS-7.1.8.zip
BASE_DIR=/path/to
FQDN=localhost
DEPLOYMENT_KEY=AMdhPkZJxwoEjwx3zV1IGOccAHNvvQ5CBVN1bk
```

```
VDdmCLA99ueRm6Cg
DEPLOYMENT_PASSWORD=password
###

CURRENT_DIR=$(pwd)

# Install a single proxy from the .zip distribution.
install() {
  echo "### Installing ${BASE_DIR}/proxy ###"
  unzip -q "${ZIP}"
  mv opendj proxy

  # The default proxyRoot is created at setup time,
but removed later.
  ./proxy/setup \
   --deploymentKey "$DEPLOYMENT_KEY" \
   --deploymentKeyPassword "$DEPLOYMENT_PASSWORD" \
   --serverId proxy \
   --rootUserDN uid=admin \
   --rootUserPassword password \
   --hostname "${FQDN}" \
   --ldapsPort 1636 \
   --adminConnectorPort 4444 \
   --profile ds-proxy-server \
   --set ds-proxy-
server/bootstrapReplicationServer:"${FQDN}:5444" \
   --set ds-proxy-server/rsConnectionSecurity:ssl \
   --set ds-proxy-server/certNickname:ssl-key-pair \
   --set ds-proxy-server/keyManagerProvider:PKCS12 \
   --set ds-proxy-server/trustManagerProvider:PKCS12
\
   --acceptLicense

   # Allow the CTS administrator to read and write
directory data.
  ./proxy/bin/dsconfig \
   create-global-access-control-policy \
    --type generic \
    --policy-name "CTS administrator access" \
    --set allowed-attribute:"*" \
    --set permission:read \
    --set permission:write \
    --set user-dn-equal-to:uid=openam_cts,ou=tokens \
    --set request-target-dn-equal-to:ou=tokens \
    --set request-target-dn-equal-to:**,ou=tokens \
```

```
    --offline \
    --no-prompt
}

# Configure distribution to multiple shards.
configure() {
  echo "### Configuring distribution for CTS shards
###"
  ./proxy/bin/dsconfig \
   create-service-discovery-mechanism \
   --mechanism-name "CTS Shard 1" \
   --type replication \
   --set bootstrap-replication-server:"${FQDN}:5444"
\
   --set bootstrap-replication-server:"${FQDN}:15444"
\
   --set ssl-cert-nickname:ssl-key-pair \
   --set key-manager-provider:PKCS12 \
   --set trust-manager-provider:PKCS12 \
   --set use-ssl:true \
   --set use-sasl-external:true \
   --offline \
   --no-prompt

  ./proxy/bin/dsconfig \
   create-service-discovery-mechanism \
   --mechanism-name "CTS Shard 2" \
   --type replication \
   --set bootstrap-replication-server:"${FQDN}:6444"
\
   --set bootstrap-replication-server:"${FQDN}:16444"
\
   --set ssl-cert-nickname:ssl-key-pair \
   --set key-manager-provider:PKCS12 \
   --set trust-manager-provider:PKCS12 \
   --set use-ssl:true \
   --set use-sasl-external:true \
   --offline \
   --no-prompt

  ./proxy/bin/dsconfig \
   create-backend \
   --backend-name distributeCts \
   --type proxy \
   --set enabled:true \
```

```
    --set partition-base-dn:ou=famrecords,ou=openam-
session,ou=tokens \
    --set shard:"CTS Shard 1" \
    --set shard:"CTS Shard 2" \
    --set use-sasl-external:true \
    --set ssl-cert-nickname:ssl-key-pair \
    --set key-manager-provider:PKCS12 \
    --set route-all:true \
    --offline \
    --no-prompt
}

delete_unused_backend() {
  echo "### Removing unused proxyRoot backend ###"
  ./proxy/bin/dsconfig \
   delete-backend \
   --backend-name proxyRoot \
   --offline \
   --no-prompt
}

cd "${BASE_DIR}" || exit 1
install
configure
delete_unused_backend
echo "### Starting proxy ###"
./proxy/bin/start-ds --quiet
cd "${CURRENT_DIR}" || exit
```

- If necessary, edit the script for use on your computer.
- Run the script.

  After the script finishes successfully, the DS proxy server is configured to distribute requests to the two partitions running on your computer.

6. Install the AM server.

   For details, see the AM Installation Guide.

   You can use the default configuration options during installation.

7. Configure the AM server to use the directory proxy server.

   For details, see Configuring External CTS Token Stores.

   This example has the following settings:

Connection string	`localhost:1636` AM must trust the DS `ssl-key-pair` server certificate to use this port.
Root suffix	`ou=famrecords,ou=openam-session,ou=tokens`
Login ID	`uid=openam_cts,ou=tokens`
Password	`password`

When you restart AM, it uses the distributed CTS store. As you perform operations in AM that use CTS, such as logging in as a regular user, you can see the token entries through the DS proxy server.

The following example shows two entries written by AM after logging in through the AM console as the `demo` user:

```
$ /path/to/proxy/bin/ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/proxy/config/keystore \
 --trustStorePassword:file
/path/to/proxy/config/keystore.pin \
 --bindDn uid=openam_cts,ou=tokens \
 --bindPassword password \
 --baseDn ou=tokens \
 "(coreTokenId=*)" \
 coreTokenId

dn: coreTokenId=id,ou=famrecords,ou=openam-
session,ou=tokens
coreTokenId: id
```

As AM performs CTS-related operations, you start to see the CTS tokens distributed across the two DS partitions. To examine the results, use LDAPS port `5636` for CTS partition 1 and `6636` for partition 2.

8. Install a second, identically configured AM server, and then test the system.

   For details, see Test Session High Availability.

9. After finishing the evaluation, stop the servers and remove the software you installed:

   - Uninstall the AM server(s).

- Tear down the DS servers:
  - Save a local copy of the script: teardown-cts.sh,

    ▼ *View script*

```
#!/usr/bin/env bash
#
# Copyright 2018-2020 ForgeRock AS. All Rights
Reserved
#
# Use of this code requires a commercial software
license with ForgeRock AS.
# or with one of its affiliates. All use shall be
exclusively subject
# to such license between the licensee and
ForgeRock AS.
#

###
# Stop and remove the proxy and replica servers.
###

###
# Adjust this path if you changed it in other
scripts:
BASE_DIR=/path/to
###

CURRENT_DIR=$(pwd)

cd "${BASE_DIR}" || exit 1
./proxy/bin/stop-ds
rm -rf proxy

for i in 5 15 6 16
do
  ./ds-rs-${i}/bin/stop-ds
done
rm -rf ds-rs-*
cd "${CURRENT_DIR}" || exit
```

  - If necessary, edit the script for use on your computer.
  - Run the script.

> After the script finishes successfully, the DS software has been removed from your computer.

On Load Balancers

A load balancer might seem like a natural component for a highly available architecture. Directory services are highly available by design, however. When used with directory services, a load balancer can do more harm than good.

The Problem

DS servers rely on data replication for high availability with tolerance for network partitions. The directory service continues to allow both read and write operations when the network is down. As a trade off, replication provides *eventual consistency*, not immediate consistency.

A load balancer configured to distribute connections or requests equitably across multiple servers can therefore *cause an application to get an inconsistent view of the directory data*. This problem arises in particular when a client application uses a pool of connections to access a directory service:

1. The load balancer directs a write request from the client application to a first server.

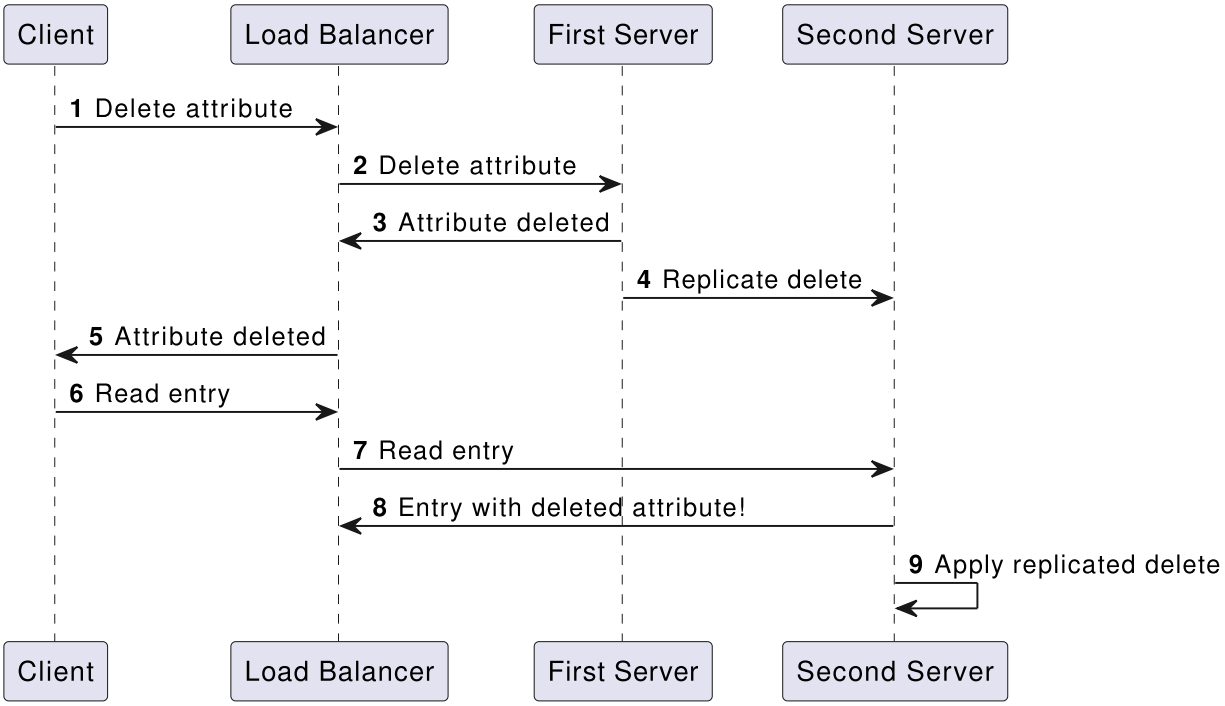
   The write request results in a change to directory data.

   The first server replicates the change to a second server, but replication is not instantaneous.
2. The load balancer directs a subsequent read request from the client application to a second server.

   The read request arrives before the change has been replicated to the second server.

   As a result, the second server returns the earlier view of the data. *The client application sees data that is different from the data it successfully changed!*

The following sequence diagram illustrates the race condition:



When used in failover mode, also known as active/passive mode, this problem is prevented. However, the load balancer adds network latency while reducing the number of directory servers actively used. This is unfortunate, since the directory server replicas are designed to work together as a pool.

Unlike many load balancers, ForgeRock Identity Platform software has the capability to account for this situation, and to balance the request load appropriately across multiple directory servers.

Recommendations

Apply the following recommendations in your directory service deployments:

Client is a ForgeRock Identity Platform 7.1 component

Do not use a load balancer between ForgeRock Identity Platform components and the DS directory service.

ForgeRock Identity Platform components use the same software as DS directory proxy to balance load appropriately across multiple directory servers.

Examples of platform components include AM and IDM.

Client opens a pool of connections to the directory service

Do not use a load balancer between the client and the DS directory service.

Configure the client application to use multiple directory servers.

Client and server are DS replicas

Never use a load balancer for replication traffic.

Client can only access a single directory server

Consider using DS directory proxy server to provide a single point of entry and balance load. Alternatively, use a load balancer in failover or active/passive mode.

Client only ever opens a single connection to the directory service
Consider using DS directory proxy server to provide a single point of entry and balance load. Alternatively, use a load balancer in failover or active/passive mode.

About Request Handling

DS servers listen for client requests using *connection handlers* . A connection handler interacts with client applications, accepting connections, reading requests, and sending responses. Most connection handlers expose configurable listen ports with security settings. The security settings point to other configuration objects, so two connection handlers can share the same certificate and private key, for example.

DS servers use different ports for different protocols. For example, a directory server might listen on port `389` for LDAP requests, port `443` for HTTPS requests, and port `4444` for administration requests from server configuration tools. Because DS servers use a different connection handler for each port, DS servers have several connection handlers enabled.

The **`setup`** command lets you initially configure connection handlers for LDAP or LDAPS, HTTP or HTTPS, and administrative traffic. The **`dsconfig`** command offers full access to all connection handler configurations.

When a client application opens a secure connection to a server, the JVM has responsibility for transport layer security negotiations. You can configure how connection handlers access keys required during the negotiations. You can also configure which clients on the network are allowed to use the connection handler. For details, see Connection Handler.

Connection handlers receive incoming requests, and pass them along for processing by the core server subsystem.

For example, an LDAP connection handler enqueues requests to the core server, which in turn requests data from the appropriate backend as necessary. For more information about backends, see Data Storage. The core server returns the LDAP response.

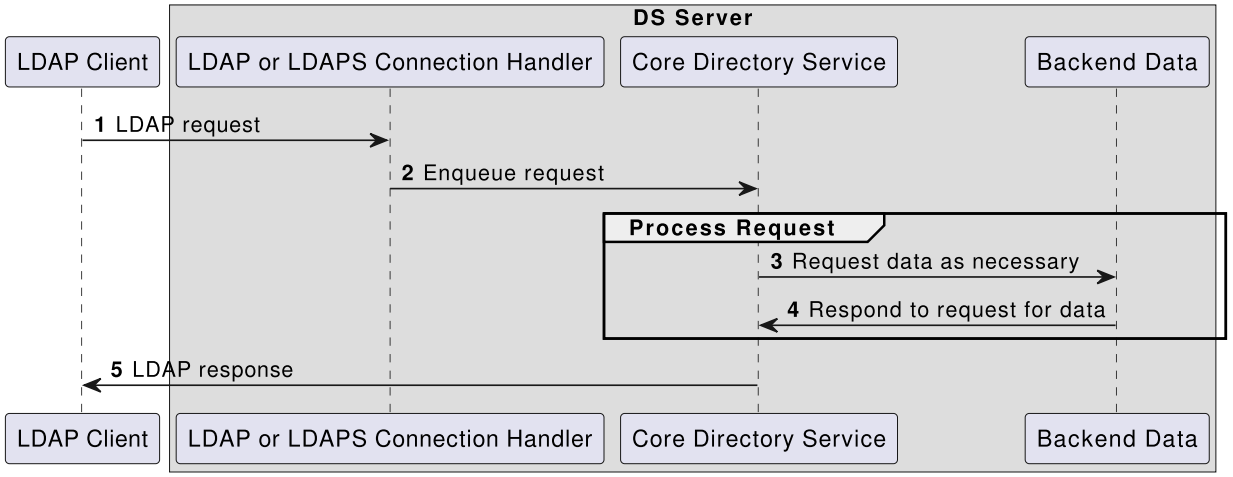


Figure 5. LDAP Requests

An HTTP connection handler translates each request to LDAP. Internally, the core server subsystem processes the resulting LDAP requests.

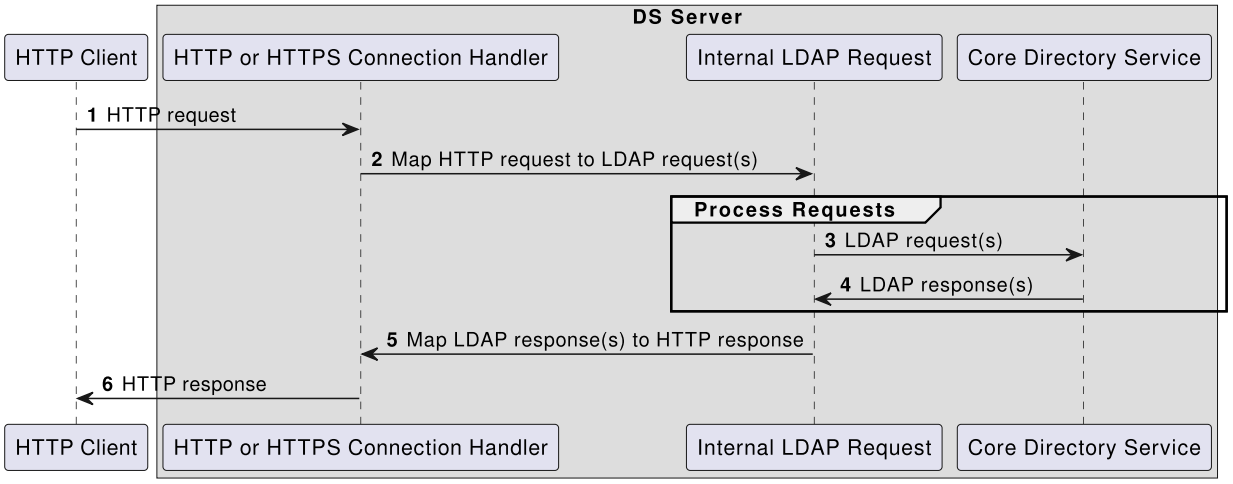


Figure 6. HTTP Requests

DS servers support other types of connection handlers. For example, JMX and SNMP connection handlers support monitoring applications. A special LDIF connection handler consumes LDIF files.

When deploying a server, decide which listen ports to expose over which networks. Determine how you want to secure the connections, as described in Secure Connections.