Installation

This guide shows you how to install and remove Directory Services software.

Evaluate DS

Try DS software.

DS for CTS

Store AM CTS tokens.

DS for Identities

Store AM identities.

DS as IDM Repo

Store IDM data.

Setup Hints

Review setup options.

DS Proxy

Install an LDAP proxy.

Component	Description
Directory Server and Tools	Pure Java, high-performance server that can be configured as: • An LDAPv3 directory server with the additional capability to serve directory data to REST applications over HTTP. • An LDAPv3 directory proxy server providing a single point of access to underlying directory servers. • A replication server handling replication traffic with directory servers and with other replication servers, receiving, sending, and storing changes to directory data. Server distributions include command-line tools for installing, configuring, and managing servers. The tools make it possible to script all operations.
DSML Gateway	DSML support is available through the gateway, which is a Java web application that you install in a web container.
REST to LDAP Gateway	In addition to the native server support for REST over HTTP, the REST to LDAP gateway is a Java web application that lets you configure REST access to any LDAPv3 directory server.
Java APIs	Java server-side APIs for server plugins that extend directory services. All Java APIs have Interface Stability: *Evolving*. *Be prepared for incompatible changes in both major and minor releases.*

Read the Release notes before installing DS software.

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.

Unpack Files

The following procedures only unpack the server files. You must then run the `setup` command to set up the server:

Unpack the Cross-Platform Zip

You can use the .zip delivery on any supported operating system.

1. Prepare for installation.
2. Unpack the cross-platform .zip file in the file system directory where you want to install the server.

   Perform this step as a user with the same file system permissions as the user who will run the **`setup`** command.

   The **`setup`** command uses the directory where you unzipped the files as the installation directory. It does not ask you where to install the server. If you want to install elsewhere on the file system, unzip the files in that location.

   Unzipping the .zip file creates a top-level `opendj` directory in the directory where you unzipped the file. On Windows systems if you unzip the file with Right-Click > Extract All, remove the trailing `opendj-`*`version`* directory from the folder you specify.

Use the Debian Package

On Debian and related Linux distributions, such as Ubuntu, you can unpack files using the Debian package:

1. Prepare for installation.

   In particular, install a Java runtime environment (JRE) if none is installed yet. The following example uses the `java11-runtime` virtual package:

   ```
   $ sudo apt-get install java11-runtime
   ```

2. Install the server package:

   ```
   $ sudo dpkg -i DS*.deb
   ```

   The Debian package:

   - Installs server files in the `/opt/opendj` directory.
   - Generates service management scripts.
   - Adds documentation files under the `/usr/share/doc/opendj` directory.
   - Adds man pages under the `/opt/opendj/share/man` directory.

The files are owned by root by default, making it easier to have the server listen on privileged ports, such as `389` and `636`.

3. Set up the server with the **`setup`** command, **`sudo /opt/opendj/setup`**.

Use the RPM Package

On Red Hat and related Linux distributions, such as Fedora and CentOS, you can unpack files using the RPM package:

1. Prepare for installation.

   In particular, install a Java runtime environment (JRE) if none is installed yet. You might need to download an RPM to install the Java runtime environment, and then install the RPM by using the **`rpm`** command:

   ```
   $ su
   Password:
   # rpm -ivh jre-*.rpm
   ```

2. Install the server package:

   ```
   # rpm -i DS*.rpm
   ```

   The RPM package:

   - Installs server files in the `/opt/opendj` directory.
   - Generates service management scripts.
   - Adds man pages under the `/opt/opendj/share/man` directory.

   The files are owned by root by default, making it easier to have the server listen on privileged ports, such as `389` and `636`.

3. Set up the server with the **`setup`** command, **`/opt/opendj/setup`**.

   By default, the server starts in run levels 2, 3, 4, and 5.

Use the Windows MSI

IMPORTANT

IMPORTANT

Prevent antivirus and intrusion detection systems from interfering with DS software.

Before using DS software with antivirus or intrusion detection software, consider the following potential problems:

Interference with normal file access

Antivirus and intrusion detection systems that perform virus scanning, sweep scanning, or deep file inspection are not compatible with DS file access, particularly write access.

Antivirus and intrusion detection software have incorrectly marked DS files as suspect to infection, because they misinterpret normal DS processing.

Prevent antivirus and intrusion detection systems from scanning DS files, except these folders:

`/path/to/opendj/bat/`

Windows command-line tools

`/path/to/opendj/bin/`

UNIX/Linux command-line tools

`/path/to/opendj/extlib/`

Optional additional `.jar` files used by custom plugins

`/path/to/opendj/lib/`

Scripts and libraries shipped with DS servers

Port blocking

Antivirus and intrusion detection software can block ports that DS uses to provide directory services.

Make sure that your software does not block the ports that DS software uses. For details, see Administrative Access.

Negative performance impact

Antivirus software consumes system resources, reducing resources available to other services including DS servers.

Running antivirus software can therefore have a significant negative impact on DS server performance. Make sure that you test and account for the performance impact of running antivirus software before deploying DS software on the same systems.

1. Prepare for installation.
2. Install the server files in one of the following ways:

a. Run the GUI installer:

   - Double-click the Windows installer package, `DS-7.1.8.msi` , to start the install wizard.
   - In the **Destination Folder** screen, set the folder location for the server files.

     The default location is under `Program Files` on the system drive. For example, if the system drive is `C:` , the default location is `C:\Program Files (x86)\opendj\` .

     The native executable is a 32-bit application, though you can run the server in a 64-bit Java environment.
   - Start the installation.

     When installation is finished, the server files are found in the location you specified.

b. Use the Microsoft **`msiexec.exe`** command to install the files.

   The following example installs the server files under `C:\opendj-7.1.8` , writing an installation log file, `install.log` , in the current folder:

   ```
   C:\> msiexec /i DS-7.1.8.msi /l* install.log /q
   OPENDJ=C:\opendj-7.1.8
   ```

Setup Hints

The following table provides extensive hints for using **`setup`** command options in the order they are presented in interactive mode, when you run the command without options.

For reference information, see setup:

Parameter	Description	Option(s)
Instance path	Server setup uses tools and templates installed with the software to generate the instance files required to run an instance of a server. By default, all the files are co-located. This parameter lets you separate the files. Set the instance path to place generated files in a different location from the tools, templates, and libraries you installed. Interactive setup suggests co-locating the software with the instance files. You cannot use a single software installation for multiple servers. Tools for starting and stopping the server process, for example, work with a single configured server. They do not have a mechanism to specify an alternate server location. If you want to set up another server, install another copy of the software, and run that copy's **`setup`** command.	`--instancePath`
Unique server ID	A server identifier string that is unique for your deployment. Choose a relatively short string, as the value is recorded repeatedly in replicated historical data.	`--serverId`

Parameter	Description	Option(s)
Deployment key	The *deployment key* is a random string generated by DS software. It is paired with a *deployment key password*, which is a random string that you choose, and that you must keep secret. Together, the deployment key and password serve to generate the shared master key that DS servers in the deployment require for protecting shared encryption secrets. By default, the `setup` command uses them to generate a private CA and keys for TLS to protect communication between DS servers. When you deploy multiple servers together, reuse the same deployment key and password for each server installation. For details, see Deployment Keys.	`--deploymentKey`
Deployment key password	This is a random string that you choose, and that you must keep secret. It is paired with the deployment key.	`--deploymentKeyPassword[:env|:file]`
Root user DN	The root user DN identifies the initial *directory superuser*. This user has privileges to perform any and all administrative operations, and is not subject to access control. It is called the root user due to the similarity to the UNIX root user. The name used in the documentation is the default name: `uid=admin`. For additional security in production environments, use a different name.	`-D, --rootUserDn`
Root user password	The root user authenticates with simple, password-based authentication. Use a strong password here unless this server is only for evaluation.	`-j, --rootUserPassword[:env|:file]`

Parameter	Description	Option(s)
Monitor user DN	The monitor user DN identifies a user with the privilege to read monitoring data (`monitor-read`). The account is replicated by default, so use the same DN on each server. The name used in the documentation is the default name: `uid=monitor` .	`--monitorUserDn`
Monitor user password	The monitor user authenticates with simple, password-based authentication. The account is replicated by default, so use the same password on each server.	`--monitorUserPassword[:env|:file]`
Fully qualified directory server domain name	The server uses the fully qualified domain name (FQDN) for identification between replicated servers. Interactive setup suggests the hostname of the local host. If this server is only for evaluation, then you can use `localhost` . Otherwise, use an FQDN that other hosts can resolve to reach your server, and that matches the FQDN in the server certificate.	`-h, --hostname`
Administration port	This is the service port used to configure the server and to run tasks. The port used in the documentation is `4444` . If the suggested port is not free, interactive setup adds 1000 to the port number and tries again, repeatedly adding 1000 until a free port is found. *Configure the firewall to allow access to this port from all connecting DS servers.*	`--adminConnectorPort`

Parameter	Description	Option(s)

Securing the deployment	Setup requires a keystore with the keys for securing connections to the administration port, and to any other secure ports you configure during setup.
	
	You can choose to use an existing keystore supported by the JVM, which can be either a file-based keystore or a PKCS#11 token. The existing keystore must protect the keystore and all private keys with the same PIN or password. If you choose a PKCS#11 token, you must first configure access through the JVM, as the only input to the **`setup`** command is the PIN.
	
	If you do not provide an existing keystore, the **`setup`** command generates a deployment key. Keep a secure copy of this key for use when installing other DS servers in your deployment. For details, see Key Management.
	
	Public key security is often misunderstood. Before making security choices for production systems, read Cryptographic Keys.
		`--useJavaKeyStore`
		`--useJceKeyStore`
		`--usePkcs11KeyStore`
		`--usePkcs12KeyStore`
		`-W, --keyStorePassword[:env|:file]`
		`-N, --certNickname`
		
		`--useJavaTrustStore`
		`--useJceTrustStore`
		`--usePkcs11TrustStore`
		`--usePkcs12TrustStore`
		`-T, --trustStorePassword[:env|:file]`

Start the server	By default, the **`setup`** command does not start the server. Finish configuring the server, then use the **`/path/to/opendj/bin/start-ds`** command. If no further configuration is required, use the **`setup --start`** option.	`-s, --start`

Parameter	Description	Option(s)

LDAP and LDAPS port	The reserved port for LDAP is `389`. The reserved port for LDAPS is `636`.
	
	Examples in the documentation use `1389` and `1636`, which are accessible to non-privileged users.
	
	If you install the server with access to privileged ports (< `1024`), and the reserved port is not yet in use, then interactive setup suggests the reserved port number. If the port is not free or cannot be used due to lack of privileges, interactive setup adds 1000 to the port number and tries again, repeatedly adding 1000 until a free port is found.
	
	The LDAP StartTLS extended operation negotiates a secure connection starting on the insecure LDAP port.
		`-p, --ldapPort`
		`-q, --enableStartTls`
		`-Z, --ldapsPort`

HTTP and HTTPS ports	The reserved port for HTTP is `80`. The reserved port for HTTPS is `443`. The interactive setup initially suggests `8080` and `8443` instead. If the initially suggested port is not free or cannot be used due to lack of privileges, interactive setup adds 1000 to the port number and tries again, repeatedly adding 1000 until a free port is found. Examples in the documentation use HTTPS on port `8443`. When you enable HTTP or HTTPS at setup time, only the administrative endpoints are enabled, `/admin/config`, `/metrics/api`, and `/metrics/prometheus`, allowing applications to configure and monitor the server. For access to user data in a directory server, see Configure HTTP User APIs.	`--httpPort` `--httpsPort`

Parameter	Description	Option(s)
Replication port	Port used for data replication messages. This port must be accessible externally from other DS servers. If this port is configured, the server acts as a replication server. It maintains a replication change log, which it exposes as an external change log by default. If the initially suggested port is not free or cannot be used due to lack of privileges, interactive setup adds 1000 to the port number and tries again, repeatedly adding 1000 until a free port is found. Examples in the documentation use `8989`.	`-r, --replicationPort`

Parameter	Description	Option(s)
Bootstrap replication servers	Specify bootstrap server `host:port` pairs, where `port` is the server's replication port. The current server contacts the bootstrap servers to discover other servers in the deployment. The `host:port` pair may represent the current server if it is a bootstrap server. Specify the same list of bootstrap servers each time you set up a replica or standalone replication server. This option interacts with the `-r, --replicationPort` option as follows: • If both options are set, the server acts as a replication server. It connects to the specified bootstrap replication server(s) to discover other servers. • If only the `-r, --replicationPort` option is set, the server acts as a replication server. It counts only itself as the bootstrap replication server. In production, specify the same list of at least two bootstrap servers every time, including when you set up the bootstrap servers. • If only the `--bootstrapReplicationServer` option is set, the server acts as a standalone directory server. It connects to the specified bootstrap replication server(s). • If neither option is set, the server is not configured for replication at setup time.	`--bootstrapReplicationServer`

Parameter	Description	Option(s)
Configure the server for use with other applications	For details, see Setup Profiles.	`--profile` `--set`

Setup Profiles

A *setup profile* lets you configure a server for a specific use case. Profiles greatly simplify the directory server setup process for such use cases, such as preparing a directory server to serve another ForgeRock® Identity Platform component product.

You can configure a setup profile using the **`setup`** command, or the **`setup-profile`** command after initial setup. The **`setup-profile`** command runs on a server that is offline.

Select a profile with the `--profile` option. Each profile has its own parameters, some of which have default values. You specify profile parameters with `--set` options.

The profile selection option takes the form `--profile` *`profileName`*`[:`*`version`*`]`. If you do not specify the optional `:`*`version`* portion of the argument, the **`setup`** command uses the current DS software version, falling back to the previous version if the current version does not match an available profile. Repeat the `--profile` option to apply multiple setup profiles.

An option to set a parameter takes the form `--set[:env|:file]` *`parameterName`*`:`*`value`* where:

- *`profileName/`* indicates which profile the parameter applies to.

  This name is required when you specify multiple profiles, and the parameter is available in more than one of the specified profiles.

  The *`profileName`* is case-insensitive.
- *`parameterName`* specifies the parameter to set.
- *`value`* specifies the value the parameter takes when the **`setup`** command applies the profile.

Use the **`setup --help-profiles`** or **`setup-profile --help`** command to list available profiles.

Use the `--help-profile` *`profileName`*`[:`*`version`*`]` option to list the parameters for the specified profile.

Check Profiles

The `opendj/profiles.version` file lists the profiles selected at setup time:

```
$ cat /path/to/opendj/config/profiles.version
ds-evaluation:7.0.0
```

Default Setup Profiles

This page lists default profiles with their parameters.

AM Configuration Data Store 6.5.0

The `am-config:6.5.0` profile has the following parameters:

`backendName`
Name of the backend for storing config
Default: `--set am-config/backendName:cfgStore`
Syntax: Name

`baseDn`
The base DN to use to store AM's configuration in
Default: `--set am-config/baseDn:ou=am-config`
Syntax: DN

`amConfigAdminPassword`
Password of the administrative account that AM uses to bind to OpenDJ
Syntax: Password

AM CTS Data Store 6.5.0

The `am-cts:6.5.0` profile has the following parameters:

`backendName`
Name of the backend for storing tokens
Default: `--set am-cts/backendName:amCts`
Syntax: Name

`baseDn`
The base DN to use to store AM's tokens in
Default: `--set am-cts/baseDn:ou=tokens`
Syntax: DN

`amCtsAdminPassword`

Password of the administrative account that AM uses to bind to OpenDJ
Syntax: Password

`tokenExpirationPolicy`
Token expiration and deletion
Default: `--set am-cts/tokenExpirationPolicy:am`

This parameter takes one of the following values:

- `am` : AM CTS reaper manages token expiration and deletion
- `am-sessions-only` : AM CTS reaper manages SESSION token expiration and deletion. DS manages expiration and deletion for all other token types. AM continues to send notifications about session expiration and timeouts to agents
- `ds` : DS manages token expiration and deletion. AM session-related functionality is impacted and notifications are not sent

AM Identity Data Store 7.1.0

The `am-identity-store:7.1.0` profile has the following parameters:

`backendName`
Name of the backend for storing identities
Default: `--set am-identity-store/backendName:amIdentityStore`
Syntax: Name

`baseDn`
The base DN to use to store identities in
Default: `--set am-identity-store/baseDn:ou=identities`
Syntax: DN

`amIdentityStoreAdminPassword`
Password of the administrative account that AM uses to bind to OpenDJ
Syntax: Password

AM Identity Data Store 7.0.0

The `am-identity-store:7.0.0` profile has the following parameters:

`backendName`
Name of the backend for storing identities
Default: `--set am-identity-store/backendName:amIdentityStore`
Syntax: Name

`baseDn`
The base DN to use to store identities in
Default: `--set am-identity-store/baseDn:ou=identities`
Syntax: DN

amIdentityStoreAdminPassword

Password of the administrative account that AM uses to bind to OpenDJ
Syntax: Password

AM Identity Data Store 6.5.0

The `am-identity-store:6.5.0` profile has the following parameters:

backendName

Name of the backend for storing identities
Default: `--set am-identity-store/backendName:amIdentityStore`
Syntax: Name

baseDn

The base DN to use to store identities in
Default: `--set am-identity-store/baseDn:ou=identities`
Syntax: DN

amIdentityStoreAdminPassword

Password of the administrative account that AM uses to bind to OpenDJ
Syntax: Password

DS Evaluation 7.0.0

The `ds-evaluation:7.0.0` profile has the following parameters:

generatedUsers

Specifies the number of generated user entries to import. The evaluation profile always imports entries used in documentation examples, such as uid=bjensen. Optional generated users have RDNs of the form uid=user.%d, yielding uid=user.0, uid=user.1, uid=user.2 and so on. All generated users have the same password, "password". Generated user entries are a good fit for performance testing with tools like addrate and searchrate
Default: `--set ds-evaluation/generatedUsers:100000`
Syntax: Number

DS Proxied Server 7.0.0

The `ds-proxied-server:7.0.0` profile has the following parameters:

proxyUserDn

The proxy user service account DN. This will be be used for authorization and auditing proxy requests.
Default: `--set ds-proxied-server/proxyUserDn:uid=proxy`
Syntax: DN

`proxyUserCertificateSubjectDn`

The subject DN of the proxy user’s certificate. The proxy must connect using mutual TLS with a TLS client certificate whose subject DN will be mapped to the proxy service account.
Default: `--set ds-proxied-server/proxyUserCertificateSubjectDn:CN=DS,O=ForgeRock.com`
Syntax: DN

`baseDn`

Base DN for user information in the server.Multiple base DNs may be provided by using this option multiple times.If no base DNs are defined then the server will allow proxying as any user, including administrator accounts.

Syntax: DN

DS Proxy Server 7.0.0

The `ds-proxy-server:7.0.0` profile has the following parameters:

`backendName`

Name of the proxy backend for storing proxy configuration
Default: `--set ds-proxy-server/backendName:proxyRoot`
Syntax: Name

`bootstrapReplicationServer`

Bootstrap replication server(s) to contact periodically in order to discover remote servers
Syntax: host:port or configuration expression

`rsConnectionSecurity`

Connection security type to use to secure communication with remote servers
Default: `--set ds-proxy-server/rsConnectionSecurity:ssl`

This parameter takes one of the following values:

- `ssl` : Use SSL
- `start-tls` : Use Start TLS

`keyManagerProvider`

Name of the key manager provider used for authenticating the proxy in mutual-TLS communications with backend server(s)
Default: `--set ds-proxy-server/keyManagerProvider:PKCS12`
Syntax: Name or configuration expression

`trustManagerProvider`

Name of the trust manager provider used for trusting backend server(s) certificate(s)
Syntax: Name or configuration expression

`certNickname`

Nickname(s) of the certificate(s) that should be sent to the server for SSL client authentication.
Default: `--set ds-proxy-server/certNickname:ssl-key-pair`
Syntax: Name or configuration expression

`primaryGroupId`

Replication domain group ID of directory server replicas to contact when available before contacting other replicas. If this option is not specified then all replicas will be treated the same (i.e all remote servers are primary)
Syntax: String or configuration expression

`baseDn`

Base DN for user information in the Proxy Server.Multiple base DNs may be provided by using this option multiple times.If no base DNs are defined then the proxy will forward requests to all public naming contexts of the remote servers
Syntax: DN or configuration expression

DS User Data Store 7.0.0

The `ds-user-data:7.0.0` profile has the following parameters:

`backendName`

Name of the backend to be created by this profile
Default: `--set ds-user-data/backendName:userData`
Syntax: Name

`baseDn`

Base DN for your users data.
Syntax: DN

`ldifFile`

Path to an LDIF file containing data to import. Use this option multiple times to specify multiple LDIF files
Syntax: File or directory path

`addBaseEntry`

Create entries for specified base DNs when the 'ldifFile' parameter is not used. When this option is set to 'false' and the 'ldifFile' parameter is not used, create an empty backend.
Default: `--set ds-user-data/addBaseEntry:true`

This parameter takes one of the following values:

- `false`
- `true`

IDM External Repository 7.1.0

The `idm-repo:7.1.0` profile has the following parameters:

`backendName`
: IDM repository backend database name
Default: `--set idm-repo/backendName:idmRepo`
Syntax: Name

`domain`
: Domain name translated to the base DN for IDM external repository data. Each domain component becomes a "dc" (domain component) of the base DN. This profile prefixes "dc=openidm" to the result. For example, the domain "example.com" translates to the base DN "dc=openidm,dc=example,dc=com".
Default: `--set idm-repo/domain:example.com`
Syntax: Domain name

IDM External Repository 7.0.0

The `idm-repo:7.0.0` profile has the following parameters:

`backendName`
: IDM repository backend database name
Default: `--set idm-repo/backendName:idmRepo`
Syntax: Name

`domain`
: Domain name translated to the base DN for IDM external repository data. Each domain component becomes a "dc" (domain component) of the base DN. This profile prefixes "dc=openidm" to the result. For example, the domain "example.com" translates to the base DN "dc=openidm,dc=example,dc=com".
Default: `--set idm-repo/domain:example.com`
Syntax: Domain name

IDM External Repository 6.5.0

The `idm-repo:6.5.0` profile has the following parameters:

`backendName`
: IDM repository backend database name
Default: `--set idm-repo/backendName:idmRepo`
Syntax: Name

`domain`
: Domain name translated to the base DN for IDM external repository data. Each domain component becomes a "dc" (domain component) of the base DN. This profile

prefixes "dc=openidm" to the result. For example, the domain "example.com" translates to the base DN "dc=openidm,dc=example,dc=com".
Default: `--set idm-repo/domain:example.com`
Syntax: Domain name

Create Your Own

If you have changes that apply to each server you set up, you can create and maintain your own setup profile.

IMPORTANT

The custom setup profile interface has stability: *Evolving*.

Be prepared to adapt your custom profiles to changes in each new release.

- Add custom setup profiles under the `opendj/template/setup-profiles/` directory.

  The **`setup`** and **`setup-profile`** commands look for profiles in that location. The default profiles provide examples that you can follow when building custom profiles.
- Add custom setup profiles after unpacking the DS files, but before running the **`setup`** or **`setup-profile`** command.
- Each setup profile strictly follows the supported file layout.

  The base path, version directories, and the `.groovy` files are required. The other files are shown here as examples:

```
opendj/template/setup-profiles/base-path/
├── version
│   ├── base-entries.ldif
│   ├── parameters.groovy
│   ├── profile.groovy
│   └── schema
│       └── schema-file-name.ldif
└── name.txt
```

File or Directory	Description
`base-path` (required)	The base path distinguishes the profile from all other profiles, and defines the profile name. Path separator characters are replaced with dashes in the name. For example, the base path `DS/evaluation` yields the profile name `ds-evaluation`.
`version` (required)	The profile version, including either two or three numbers. Numbers can be separated by dots (`.`) or dashes (`-`). Set this to the version of the software that the profile is for. For example, if you are writing a profile for Transmogrifier 2.0, use `2.0`. Add multiple versions of your profile when using the same DS version with different versions of your application.
`base-entries.ldif`	Optional LDIF file that templates the entries this profile adds to the directory. This is an example of a file used by `profile.groovy`.
`parameters.groovy` (required)	Groovy script defining profile parameters that users can set. See Parameters.
`profile.groovy` (required)	Groovy script that makes changes to the server. See Profile Script.
`schema-file-name.ldif`	Optional LDAP schema file required for entries added by this profile. This is an example of a file used by `profile.groovy`.
`name.txt`	If this file is present, it must hold the human-readable form of the profile name, *not including the version*, in a single-line text file.

At setup time, the user cannot select more than one version of the same setup profile. The user can select multiple setup profiles for the same server. You must ensure that your profile is not incompatible with other available profiles.

Parameters

You let users set parameters through the `parameters.groovy` script. The profile uses the parameters as variables in the `profile.groovy` script, and resource files.

The `parameters.groovy` script lists all parameter definitions for the profile. It includes only parameter definitions. Each parameter definition is resolved at runtime, and so must not be provided programmatically. Parameter definitions start with `define`, and can have the following methods:

```
define.type "name"                                  \
  advanced()                                        \
  defaultValueFromSetupTool global-setup-option     \
  defaultValue default                              \
  description "short-description"                   \
  help "help-message"                               \
  multivalued()                                     \
  multivalued "help message(s)"                     \
  optional()                                        \
  optional "help message(s)"                        \
  descriptionIfNoValueSet "short-description"       \
  property "property-name"                          \
  prompt "prompt message(s)"                        \
  expressionAllowed()
```

Element	Description
`type` (required)	This mandatory parameter type is one of the following. The profile mechanism converts the string input internally into a Java class: • `booleanParameter` • `dnParameter` (a DN) • `domainParameter` The input is a domain that the profile mechanism converts to a DN. The domain `example.com` becomes `dc=example,dc=com`. • `hostPortParameter` (a hostname:port pair) • `doubleParameter` (a Double number) • `floatParameter` (a Float number) • `integerParameter` (an Integer number) • `longParameter` (a Long number) • `passwordParameter` The input is a password, and encoded with a password storage scheme to avoid exposing the plain text. • `pathParameter` • `stringParameter`
`name` (required)	This mandatory parameter name must be a valid Groovy name string.
`advanced()`	This is an advanced parameter, meaning interactive mode does not show the parameter or prompt for input. When using `advanced()`, you must set a default parameter value. Interactive mode sets this parameter to the default value.

Element	Description
`defaultValueFromSetupTool` *`global-setup-option`*	This parameter takes its default from the value of the specified the global **`setup`** option. The `defaultValueFromSetupTool` method only applies when the profile is used by the **`setup`** command. The *global-setup-option* is the option name without leading dashes.
`defaultValue` *`default`*	This parameter's default must match the type.
`description "`*`short-description`*`"`	This provides a brief summary of what the parameter does. The "short-description" is a single paragraph with no trailing punctuation.
`help "`*`help-message`*`"`	The message, used in online help, provides further explanation on how to use the parameter. The "help-message" is a single paragraph with no trailing punctuation.
`multivalued()`	This parameter takes multiple values, and no help message is required. Use this, for example, when the property is `advanced()`.
`multivalued "`*`help message(s)`*`"`	This parameter takes multiple values, and interactive mode prompts the user for each one. Each help message string is a single paragraph, and the final help message has no trailing punctuation. Help message arguments are separated with commas.
`optional()`	This parameter is optional, and no help message is required.

Element	Description
`optional "`*`help message(s)`*`"`	This parameter is optional, and interactive mode prompts the user for input. Each help message string is a single paragraph, and the final help message has no trailing punctuation. Help message arguments are separated with commas.
`descriptionIfNoValueSet "`*`short-description`*`"`	The description is displayed when the parameter is optional, and the user has not set a value. The "short-description" is a single paragraph with no trailing punctuation.
`property "`*`property-name`*`"`	The profile replaces `&{`*`property-name`*`}` in resource files with the value of this property. The `&{`*`property-name`*`}` expressions follow the rules described in Property Value Substitution.
`prompt "`*`prompt message(s)`*`"`	In interactive mode, display one or more paragraphs when prompting the user for input. Each prompt message string is a single paragraph. If no default value is set the final prompt message takes a trailing colon. Prompt message arguments are separated with commas.

Profile Script

When a user requests a profile, the `profile.groovy` script controls what the profile does.

When developing your profile script, you can use these classes and methods, which are bound into the execution context of your script before it executes:

In addition, see the Javadoc for the setup model.

Default Imports

The profile mechanism imports the following classes and methods, making them available by default in the context of your profile script:

```
import static org.forgerock.i18n.LocalizableMessage.raw
import static
```

```
org.forgerock.opendj.setup.model.Profile.ParameterType.of
import static java.nio.file.Files.*

import org.forgerock.i18n.LocalizableMessage
import org.forgerock.opendj.ldap.Dn
import org.forgerock.opendj.setup.model.SetupException

import java.nio.file.Paths
```

Server Methods

A `ds` object is bound into the execution context of your profile script.

All its methods throw a `SetupException` on failure. On failure, the setup process removes the server's `db` and `config` directories. This allows the user to start over, applying the same profiles again.

All the `ds` methods run with the server offline:

Method	Description
`void ds.addBackend(String backendName, String entryDn)`	Creates the backend, adds it to the server, and sets it as the working backend. When you use other methods to import LDIF and create indexes, they operate on the working backend. Use `importBaseEntry` to add only the base entry, or `importLdif` to add entries to the backend.
`void ds.addBackend(String backendName, Dn entryDn)`	
`void ds.addBackendWithDefaultUserIndexes(String backendName, String entryDn)`	
`void ds.addBackendWithDefaultUserIndexes(String backendName, Dn entryDn)`	
`void ds.setWorkingBackend(String backendName)`	Set the specified backend as the one to operate on when importing LDIF and creating indexes.

Method	Description
`void ds.importBaseEntry(Dn baseDn)`	Import the entry with the specified base DN as the base entry of the working backend. The import operation erases any previous content of the backend before importing new content.
`void ds.importLdifWithSampleEntries(Dn sampleEntryBaseDn, int nbSampleEntries, String… ldifFilePaths)`	Import the specified number of sample entries with the specified base DN, based on the LDIF file templates provided, to the working backend. The LDIF must contain the base entry. This method replaces `&{property-name}` in the LDIF with the property values before import. The import operation erases any previous content of the backend before importing new content.
`void ds.importLdifTemplate(String… ldifFilePaths)`	Add the entries from the LDIF files provided to the working backend. The LDIF must contain the base entry. This method replaces `&{property-name}` in the LDIF with the property values before import. The import operation erases any previous content of the backend before importing new content.
`void ds.importLdifTemplate(Collection<Path> ldifFilePaths)`	
`void ds.importLdif(String… ldifFilePaths)`	Add the entries from the LDIF files provided to the working backend. The LDIF must contain the base entry. If there are multiple files, each entry must appear only once. The import operation erases any previous content of the backend before importing new content.
`void ds.importLdif(Collection<Path> ldifFilePaths)`	

Method	Description
`void ds.addSchemaFiles()`	Copy LDIF-format schema files from the `schema` directory of the profile to the `db/schema` directory of the server. If no `schema` directory is present for the current version of the profile, this method uses schema from a previous version of the profile.
`void ds.config(List<String> cliArgs)`	Run the **`dsconfig`** command in offline mode with the specified arguments. Use this method only for additional configuration, not when creating backends or indexes.
`void ds.addIndex(String attributeName, String… types)`	Create indexes of the specified types in the working backend for the specified attribute. For a list of available index types, see index-type.
`void ds.failSetup(String message)`	Cause the profile to fail with a `SetupException` having the specified message.

Resource File Methods

A `resource` object is bound into the execution context of your profile script. The `resource` methods let you retrieve arbitrary files from the profile, and replace configuration expressions in resource files:

Method	Description
`Path resource.file(String path)`	Return the path to the specified resource file. If the specified path is relative, the method first returns the path of the file in the current profile version, or the path of the file in the previous profile version if none is present in the current version. If the specified path is absolute, the method only converts the string to a path.
`void resource.applyTemplate(String template, String target)`	Convert the relative template path as `resource.file(template)` ,the relative target path as an absolute path, and copy the template file to the target file, replacing `&{`*`property-name`*`}` with the property values. The `&{`*`property-name`*`}` expressions follow the rules described in Property Value Substitution.

Logging Methods

Use the `console` object to write log messages when your profile script runs.

The `console` object implements SetupConsole, and so provides all the methods documented for that interface.

The **`setup`** and **`setup-profile`** commands log any exceptions that occur when your profile script runs, so there is no need to catch exceptions just for logging purposes.

Install DS for Evaluation

▼ More information

To set up the server, use the setup command-line tool.

When used without options, the command is interactive.

The following **`setup`** options are mandatory. When performing a non-interactive, silent installation, specify at least all mandatory options as part of the command. If you use only these options, the command sets up a server listening only on an administration port. The administration port is protected by a key pair specified or generated at setup time:

- `--adminConnectorPort {port}` (conventional port number: `4444`)
- `--hostname {hostname}`
- `--rootUserDN {rootUserDN}` (default: `uid=admin`)
- `--rootUserPassword {rootUserPassword}`

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Generate a deployment key unless you already have one:

   ```
   $ dskeymgr create-deployment-key --deploymentKeyPassword
   password

   your-deployment-key
   ```

   Save the deployment key and its deployment password. Keep the key and the password safe, and keep the password secret. Use the same deployment key and password for all the servers in the same environment.

   ▼ ***More about deployment keys***

   A *deployment key* is a random string generated by DS software. A *deployment key password* is a secret string at least 8 characters long that you choose. The two are a pair. You must have a deployment key's password to use the key.

   Each deployment requires a *single, unique deployment key and its password*. DS uses the pair to:

   - Protect the keys to encrypt and decrypt backup files and directory data.
   - Generate the TLS key pairs to protect secure connections, unless you provide your own.

   Store your deployment key and password in a safe place, and reuse them when configuring other servers in the same deployment.

   The DS `setup` and `dskeymgr` commands use the pair to generate the following:

   - (Required) A shared master key for the deployment.

     DS replicas share secret keys for data encryption and decryption. DS servers encrypt backend data, backup files, and passwords, and each replica must be able to decrypt data encrypted on another peer replica.

    To avoid exposing secret keys, DS servers encrypt secret keys with a shared master key. DS software uses a deployment key and its password to derive the master key.

  - (Optional) A private PKI for trusted, secure connections.

    A PKI serves to secure network connections from clients and other DS servers. The PKI is a trust network, requiring trust in the CA that signs public key certificates.

    Building a PKI can be complex. You can use self-signed certificates, but you must distribute each certificate to each server and client application. You can pay an existing CA to sign certificates, but that has a cost, and leaves control of trust with a third party. You can set up a CA or certificate management software, but that can be a significant effort and cost. As a shortcut to setting up a private CA, DS software uses deployment keys and passwords.

    DS software uses the deployment key and its password to generate key pairs without storing the CA private key.

  For additional details, see Deployment Keys.

3. Set the deployment key as the value of the environment variable, `DEPLOYMENT_KEY`:

```
$ export DEPLOYMENT_KEY=your-deployment-key
```

   Examples in the documentation show this environment variable as a reminder to use your own key. Other options are available, as described by the **`setup --help`** command.

4. Run the **`setup`** command to install a directory server replica with the evaluation profile:

```
# Set up a directory server for evaluation.
$ /path/to/opendj/setup \
 --serverId evaluation-only \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword password \
 --monitorUserPassword password \
 --hostname localhost \
 --adminConnectorPort 4444 \
 --ldapPort 1389 \
 --enableStartTls \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --replicationPort 8989 \
 --bootstrapReplicationServer localhost:8989 \
 --profile ds-evaluation \
 --start \
 --acceptLicense
```

▼ *More information*

- The **`setup`** command is located where you installed the files.
- The setup process uses the deployment key you generated, and its password.

  If you specify only a deployment password, and no deployment key, the **`setup`** command generates a deployment key and displays it in the output.
- This example prepares a single server for evaluation, so the hostname is `localhost`.

  In production, use fully qualified domain names, such as `ds.example.com`.
- The server is ready to replicate sample data with other servers, but there are no other replicas, yet.

  For now, the server points to itself as a bootstrap replication server. To get started with replication, see Learn Replication.
- It sets a password for the default monitoring user account, `uid=Monitor`.
- The server listens for requests on the ports used in examples throughout the documentation.
- For evaluation purposes, no further configuration is required.

  The `--start` option forces the server to start as part of the setup process.

Learn About the Evaluation Setup Profile

The evaluation setup profile helps you learn and demonstrate directory services. Unlike other setup profiles, which use secure, production-ready access control settings, the evaluation setup profile provides easy access to sample data with the following features:

- Sample Example.com data.

The sample data has the base DN `dc=example,dc=com`. It includes more than 100 hand-written entries for users, groups, and devices.

By default, it also includes 100,000 generated users, with DNs from `uid=user.0,ou=people,dc=example.dc=com` to `uid=user.99999,ou=people,dc=example.dc=com`.

Use the `--set ds-evaluation/generatedUsers:`*`number`* option to generate a different number of additional entries. Each generated user has the same password, which is `password`.

The hand-written sample Example.com data includes a group of directory administrators, `cn=Directory Administrators,ou=Groups,dc=example,dc=com`. Members of this group, such as `kvaughan`, have full access to directory data.

Examples throughout the documentation demonstrate features using this sample data.

- Global permission to perform operations over insecure connections.
- REST to LDAP enabled by default.
- Additional schema for examples demonstrating class of service and JSON attributes.
- Custom matching rule providers for JSON attributes.
- Many permissions, such as anonymous read and search access, listed in the table that follows.

The evaluation setup profile lets you learn and demonstrate most directory features without adding any ACIs.

Name	Description	ACI Definition
Anonymous extended operation access	Anonymous and authenticated users can request the LDAP extended operations that are specified by OID or alias. Modification or removal may affect applications.	`(targetcontrol="Assertion||AuthorizationIdentity||MatchedValues||NoOp||PasswordPolicy||PasswordQualityAdvice||PermissiveModify||PostRead||PreRead||RealAttrsOnly||SimplePagedResults||TransactionId||VirtualAttrsOnly||Vlv") (version 3.0; acl "Anonymous extended operation access"; allow(read) userdn="ldap:///anyone";)`
Anonymous extended operation access	Anonymous and authenticated users can request the LDAP extended operations that are specified by OID or alias. Modification or removal may affect applications.	`(extop="Cancel||GetSymmetricKey||PasswordModify||StartTls||WhoAmI") (version 3.0; acl "Anonymous extended operation access"; allow(read) userdn="ldap:///anyone";)`
Anonymous read and search access	Anonymous and authenticated Example.com users can read the user data attributes that are specified by their names.	`(targetattr!="userPassword||authPassword||debugsearchindex") (version 3.0; acl "Anonymous read and search access"; allow (read,search,compare) userdn="ldap:///anyone";)`

Name	Description	ACI Definition
Authenticated control use	Authenticated Example.com users can proxy and examine CSNs.	`(targetcontrol="ProxiedAuth||Csn") (version 3.0; acl "Authenticated control use"; allow(read) userdn="ldap:///all"; )`
Authenticated users extended operation access	Authenticated users can request the LDAP extended operations that are specified by OID or alias. Modification or removal may affect applications.	`(targetcontrol="ManageDsaIt||RelaxRules||ServerSideSort||SubEntries||SubtreeDelete") (version 3.0; acl "Authenticated users extended operation access"; allow(read) userdn="ldap:///all"; )`
Authenticated users extended operation access	Authenticated users can request the LDAP extended operations that are specified by OID or alias. Modification or removal may affect applications.	`(extop="PasswordPolicyState") (version 3.0; acl "Authenticated users extended operation access"; allow(read) userdn="ldap:///all"; )`
Directory administrator full access	Example.com directory administrators have access to read and write Example.com directory data, rename and move entries, and use proxied authorization.	`(targetattr="*") (version 3.0; acl "Directory administrator full access"; allow (all,export,import,proxy) groupdn="ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com"; )`

Name	Description	ACI Definition
Proxied authorization for apps	Example.com applications can make requests on behalf of other users.	`(targetattr="*") (version 3.0; acl "Proxied authorization for apps"; allow (all,proxy) (userdn="ldap:///cn=*,ou=Apps,dc=example,dc=com");)`
Self entry modification	Authenticated users can modify the specified attributes on their own entries.	`(targetattr=" audio || authPassword || description || displayName || givenName || homePhone || homePostalAddress || initials || jpegPhoto || labeledURI || mobile || pager || postalAddress || postalCode || preferredLanguage || telephoneNumber || userPassword") (version 3.0; acl "Self entry modification"; allow (write) userdn="ldap:///self";)`
Self entry read for passwords	Authenticated users can read the password values on their own entries. By default, the server applies a one-way hash algorithm to the password value before writing it to the entry, so it is computationally difficult to recover the plaintext version of the password from the stored value.	`(targetattr="userPassword||authPassword") (version 3.0; acl "Self entry read for passwords"; allow (read,search,compare) userdn="ldap:///self";)`

Name	Description	ACI Definition
Self service group creation	Authenticated Example.com users can create self service groups.	`(targattrfilters="add=objectClass:(objectClass=groupOfNames)")(version 3.0; acl "Self service group creation";allow (add)(userdn="ldap:///uid=*,ou=People,dc=example,dc=com");)`
Self service group deletion	The authenticated owner of a self service group can delete the group.	`(version 3.0; acl "Self service group deletion";allow (delete)(userattr="owner#USERDN");)`
Self service group registration	Authenticated Example.com users can sign themselves up as members of self service groups.	`(targetattr="member")(version 3.0; acl "Self service group registration"; allow (selfwrite)(userdn="ldap:///uid=*,ou=People,dc=example,dc=com");)`
User-Visible Monitor Attributes	Authenticated users can read monitoring information if they have the monitor read privilege. Modification or removal may affect applications.	`(target="ldap:///cn=monitor")(targetattr="*||+")(version 3.0; acl "User-Visible Monitor Attributes"; allow (read,search,compare) userdn="ldap:///all";)`

Name	Description	ACI Definition
User-visible operational attributes	Anonymous and authenticated users can read attributes that identify entries and that contain information about modifications to entries.	`(targetattr=" createTimestamp || creatorsName || modifiersName || modifyTimestamp || entryDN || entryUUID || subschemaSubentry || etag || governingStructureRule || structuralObjectClass || hasSubordinates || numSubordinates || isMemberOf") (version 3.0; acl "User-visible operational attributes"; allow (read,search,compare) userdn="ldap:///anyone ";)`

Name	Description	ACI Definition
User-Visible Root DSE Operational Attributes	Anonymous and authenticated users can read attributes that describe what the server supports. Modification or removal may affect applications.	`(target="ldap:///") (targetscope="base") (targetattr="objectClass||namingContexts||subSchemaSubEntry||supportedAuthPasswordSchemes||supportedControl||supportedExtension||supportedFeatures||supportedLDAPVersion||supportedSASLMechanisms||supportedTLSCiphers||supportedTLSProtocols||vendorName||vendorVersion||fullVendorVersion||alive||healthy") (version 3.0; acl "User-Visible Root DSE Operational Attributes"; allow (read,search,compare) userdn="ldap:///anyone";)`
User-Visible Schema Operational Attributes	Authenticated users can read LDAP schema definitions. Modification or removal may affect applications.	`(target="ldap:///cn=schema") (targetscope="base") (targetattr="objectClass||attributeTypes||dITContentRules||dITStructureRules ||ldapSyntaxes||matchingRules||matchingRuleUse||nameForms||objectClasses||etag||modifiersName||modifyTimestamp") (version 3.0; acl "User-Visible Schema Operational Attributes"; allow (read,search,compare) userdn="ldap:///all";)`

Install DS for AM CTS

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Run the appropriate **`setup`** command with the `--profile am-cts` option.

   Installation settings depend on AM token expiration and session capability requirements:

   a. AM reaper manages all token expiration:

   ```
   $ /path/to/opendj/setup \
    --deploymentKey $DEPLOYMENT_KEY \
    --deploymentKeyPassword password \
    --rootUserDN uid=admin \
    --rootUserPassword str0ngAdm1nPa55word \
    --monitorUserPassword str0ngMon1torPa55word \
    --hostname ds.example.com \
    --adminConnectorPort 4444 \
    --ldapPort 1389 \
    --enableStartTls \
    --ldapsPort 1636 \
    --httpsPort 8443 \
    --replicationPort 8989 \
    --bootstrapReplicationServer rs1.example.com:8989 \
    --bootstrapReplicationServer rs2.example.com:8989 \
    --profile am-cts \
    --set am-cts/amCtsAdminPassword:5up35tr0ng \
    --acceptLicense
   ```

   b. AM reaper manages only SESSION token expiration:

   ```
   $ /path/to/opendj/setup \
    --deploymentKey $DEPLOYMENT_KEY \
    --deploymentKeyPassword password \
    --rootUserDN uid=admin \
    --rootUserPassword str0ngAdm1nPa55word \
    --monitorUserPassword str0ngMon1torPa55word \
    --hostname ds.example.com \
    --adminConnectorPort 4444 \
    --ldapPort 1389 \
    --enableStartTls \
    --ldapsPort 1636 \
   ```

```
 --httpsPort 8443 \
 --replicationPort 8989 \
 --bootstrapReplicationServer rs1.example.com:8989 \
 --bootstrapReplicationServer rs2.example.com:8989 \
 --profile am-cts \
 --set am-cts/amCtsAdminPassword:5up35tr0ng \
 --set am-cts/tokenExpirationPolicy:am-sessions-only \
 --acceptLicense
```

c. DS manages all token expiration:

```
$ /path/to/opendj/setup \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword str0ngAdm1nPa55word \
 --monitorUserPassword str0ngMon1torPa55word \
 --hostname ds.example.com \
 --adminConnectorPort 4444 \
 --ldapPort 1389 \
 --enableStartTls \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --replicationPort 8989 \
 --bootstrapReplicationServer rs1.example.com:8989 \
 --bootstrapReplicationServer rs2.example.com:8989 \
 --profile am-cts \
 --set am-cts/amCtsAdminPassword:5up35tr0ng \
 --set am-cts/tokenExpirationPolicy:ds \
 --acceptLicense
```

For details about the mechanism DS uses to expire tokens, see Entry Expiration.

In each of the preceding example commands:

- The deployment key for installing the server is stored in the environment variable `DEPLOYMENT_KEY`. Install all servers in the same deployment with the same deployment key and deployment key password. For details, read Key Management.
- The service account to use in AM when connecting to DS has:
  - Bind DN: `uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens`.

        - Password: The password you set with `am-cts/amCtsAdminPassword`.
    - The base DN for AM CTS tokens is `ou=famrecords,ou=openam-session,ou=tokens`.
    - The `am-cts` profile excludes the base DN from change number indexing.

   For the full list of profiles and parameters, see Default Setup Profiles.

3. Finish configuring the server *before you start it*.

   For a list of optional steps at this stage, see Install DS for Custom Cases.

4. Start the server:

```
$ /path/to/opendj/bin/start-ds
```

Install DS for AM Configuration

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Run the **setup** command with the `--profile am-config` option:

```
$ /path/to/opendj/setup \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword str0ngAdm1nPa55word \
 --monitorUserPassword str0ngMon1torPa55word \
 --hostname ds.example.com \
 --adminConnectorPort 4444 \
 --ldapPort 1389 \
 --enableStartTls \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --replicationPort 8989 \
 --bootstrapReplicationServer rs1.example.com:8989 \
 --bootstrapReplicationServer rs2.example.com:8989 \
 --profile am-config \
 --set am-config/amConfigAdminPassword:5up35tr0ng \
 --acceptLicense
```

    - The deployment key for installing the server is stored in the environment variable `DEPLOYMENT_KEY`. Install all servers in the same deployment with the same deployment key and deployment key password. For details, read Key Management.
    - The service account to use in AM when connecting to DS has:
        - Bind DN: `uid=am-config,ou=admins,ou=am-config`.
        - Password: The password you set with `am-config/amConfigAdminPassword`.
    - The base DN for AM configuration data is `ou=am-config`.
    - AM does not require change number indexing, which involves resource-intensive processing. If AM is the only application using this data, disable change number indexing. For details, see Disable Change Number Indexing.

   For the full list of profiles and parameters, see Default Setup Profiles.

3. Finish configuring the server *before you start it*.

   For a list of optional steps at this stage, see Install DS for Custom Cases.

4. Start the server:

```
$ /path/to/opendj/bin/start-ds
```

Install DS for AM Identities

Use this profile when setting up DS as an AM identity repository and user data store. It includes the additional LDAP schema and indexes required to store AM identities:

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Run the **`setup`** command with the `--profile am-identity-store` option:

```
$ /path/to/opendj/setup \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword str0ngAdm1nPa55word \
 --monitorUserPassword str0ngMon1torPa55word \
 --hostname ds.example.com \
 --adminConnectorPort 4444 \
 --ldapPort 1389 \
 --enableStartTls \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --replicationPort 8989 \
 --bootstrapReplicationServer rs1.example.com:8989 \
 --bootstrapReplicationServer rs2.example.com:8989 \
 --profile am-identity-store \
 --set am-identity-
store/amIdentityStoreAdminPassword:5up35tr0ng \
 --acceptLicense
```

- The deployment key for installing the server is stored in the environment variable `DEPLOYMENT_KEY`. Install all servers in the same deployment with the same deployment key and deployment key password. For details, read Key Management.
- The service account to use in AM when connecting to DS has:
  - Bind DN: `uid=am-identity-bind-account,ou=admins,ou=identities`.
  - Password: The password you set with `am-identity-store/amIdentityStoreAdminPassword`.
- The base DN for AM identities is `ou=identities`.
- AM does not require change number indexing, which involves resource-intensive processing. If AM is the only application using this data, disable change number indexing. For details, see Disable Change Number Indexing.

  For the full list of profiles and parameters, see Default Setup Profiles.

3. Finish configuring the server *before you start it*.

   For a list of optional steps at this stage, see Install DS for Custom Cases.

4. Start the server:

```
$ /path/to/opendj/bin/start-ds
```

Install DS as an IDM Repository

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Run the **setup** command with the `--profile idm-repo` option:

   ```
   $ /path/to/opendj/setup \
    --deploymentKey $DEPLOYMENT_KEY \
    --deploymentKeyPassword password \
    --rootUserDN uid=admin \
    --rootUserPassword str0ngAdm1nPa55word \
    --hostname localhost \
    --adminConnectorPort 34444 \
    --ldapPort 31389 \
    --enableStartTls \
    --profile idm-repo \
    --set idm-repo/domain:forgerock.com \
    --acceptLicense
   ```

   - The deployment key for installing the server is stored in the environment variable `DEPLOYMENT_KEY`. Install all servers in the same deployment with the same deployment key and deployment key password. For details, read Key Management.
   - The administrative account to use in IDM when connecting to DS has:
     - Bind DN: The DN set with the `--rootUserDN` option.
     - Password: The password set with the `--rootUserPassword` option.
   - The base DN for IDM data is `dc=openidm,dc=forgerock,dc=com`.
   - IDM requires change number indexing with the default settings.

   For the full list of profiles and parameters, see Default Setup Profiles.

3. Finish configuring the server *before you start it*.

   For a list of optional steps at this stage, see Install DS for Custom Cases.

4. If all access to DS goes through IDM, IDM manages password policy.

   In this case, relax the default password policy settings:

   ```
   $ dsconfig \
    set-password-policy-prop \
    --policy-name "Default Password Policy" \
    --reset password-validator \
    --offline \
    --no-prompt
   ```

```
$ dsconfig \
 set-password-policy-prop \
 --policy-name "Root Password Policy" \
 --reset password-validator \
 --offline \
 --no-prompt
```

5. Start the server:

```
$ /path/to/opendj/bin/start-ds
```

Install DS for User Data

This profile includes indexes for `inetOrgPerson` entries. It is not intended for deployments with AM or IDM identities.

It does not include the additional LDAP schema and indexes required to store AM identities. To set up a user data store for AM or for sharing between AM and IDM, see Install DS for AM Identities instead.

To import generated sample user data, see Install DS for Evaluation instead:

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Run the **setup** command with the `--profile ds-user-data` option:

```
$ /path/to/opendj/setup \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword str0ngAdm1nPa55word \
 --monitorUserPassword str0ngMon1torPa55word \
 --hostname ds.example.com \
 --adminConnectorPort 4444 \
 --ldapPort 1389 \
 --enableStartTls \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --replicationPort 8989 \
 --bootstrapReplicationServer rs1.example.com:8989 \
 --bootstrapReplicationServer rs2.example.com:8989 \
 --profile ds-user-data \
 --set ds-user-data/baseDn:dc=example,dc=com \
 --set ds-user-data/ldifFile:/tmp/user-data.ldif \
 --acceptLicense
```

In this example, the `/tmp/user-data.ldif` file contains the user data entries to import. This is just a placeholder. When you run the command, replace it with your LDIF file containing your own user data.

- The deployment key for installing the server is stored in the environment variable `DEPLOYMENT_KEY`. Install all servers in the same deployment with the same deployment key and deployment key password. For details, read Key Management.
- The data is stored in the `userData` backend.

For the full list of profiles and parameters, see Default Setup Profiles.

3. Finish configuring the server *before you start it*.

   For a list of optional steps at this stage, see Install DS for Custom Cases.

4. Start the server:

```
$ /path/to/opendj/bin/start-ds
```

This setup profile creates the following indexes for user data:

Index	Approx.	Equality	Ordering	Presence	Substring	Entry Limit
`aci`	-	-	-	Yes	-	4000
`cn`	-	Yes	-	-	Yes	4000
`dn2id`	Non-configurable internal index					
`ds-certificate-fingerprint`	-	Yes	-	-	-	4000
`ds-certificate-subject-dn`	-	Yes	-	-	-	4000
`ds-sync-conflict`	-	Yes	-	-	-	4000

Index	Approx.	Equality	Ordering	Presence	Substring	Entry Limit
`ds-sync-hist`	-	-	Yes	-	-	4000
`entryUUID`	-	Yes	-	-	-	4000
`givenName`	-	Yes	-	-	Yes	4000
`id2children`	Non-configurable internal index					
`id2subtree`	Non-configurable internal index					
`mail`	-	Yes	-	-	Yes	4000
`member`	-	Yes	-	-	-	4000
`objectClass`	-	Yes	-	-	-	4000
`sn`	-	Yes	-	-	Yes	4000
`telephoneNumber`	-	Yes	-	-	Yes	4000
`uid`	-	Yes	-	-	-	4000
`uniqueMember`	-	Yes	-	-	-	4000

Install DS for Custom Cases

Follow these steps to install a DS replica with your own custom configuration:

1. Before proceeding, install the server files.

   For details, see Unpack Files.
2. Run the **`setup`** command with any required setup profiles.
3. Finish configuring the server.

   Perform any of the following optional steps *before starting the server*.

   Use the `--offline` option with commands instead of the credentials and connection information shown in many examples:

   - Add custom syntaxes and matching rules.

     For examples, see Indexes for JSON.
   - Configure password storage.

For details, see Configure Password Policies.

Take care to configure the password policy import plugin as well. For details on the settings, see Password Policy Import Plugin.

- Add custom LDAP schema.

  For details, see LDAP Schema.

- Configure one or more backends for your data.

  For details, see Create a Backend. When you create the backend, unless you choose *not* to replicate the data, follow each step of the procedure, adapting the example commands for offline use:

  - Configure the new backend using the `dsconfig create-backend` as shown.
  - Verify that replication is enabled using the `dsconfig get-synchronization-provider-prop` command as shown.
  - Let the server replicate the base DN of the new backend, using the `dsconfig create-replication-domain` command as shown to configure the replication domain.
  - If you have existing data for the backend, make appropriate plans to initialize replication, as described in Manual Initialization.

- Configure indexes for the backends you configured.

  For details, see Indexes.

- Make sure the server has the shared master key for encrypted data and backups.

  If you set up the servers with a known deployment key and password, you have nothing to do.

  If you do not know the deployment key and password, see Replace Deployment Keys.

- Initialize replication.

  For example, import the data from LDIF, or restore the data from backup.

  For details, see Manual Initialization, Import LDIF, or Restore.

4. Start the server:

```
$ /path/to/opendj/bin/start-ds
```

When you start the server, it generates initial state identifiers (generation IDs) for its replicated base DNs. If you perform the above configuration steps on replicas separately after starting them, their generation IDs can be out of sync.

When generation IDs do not match on different replicas for a particular base DN, DS must assume that the replicas do not have the same data. As a result, replication cannot proceed. To fix the mismatch of this replica's generation IDs with other replicas, stop the server and clear all replication data:

```
$ /path/to/opendj/bin/stop-ds
$ /path/to/opendj/bin/dsrepl clear-changelog
```

IMPORTANT

Clearing the changelog before all the changes have been sent to other replication servers can cause you to lose data.

Use the **`dsrepl clear-changelog`** command only when initially setting up the replica, unless specifically instructed to do so by a qualified ForgeRock technical support engineer.

Complete any further configuration necessary while the replica is stopped to align it with other replicas. When you start the replica again with the **`start-ds`** command, other replication servers update it with the data needed to resume replication.

For details on replication, see Replication.

Install Directory Proxy

Directory proxy servers forward LDAP requests for user data to remote directory servers. Proxy servers make it possible to provide a single point of access to a directory service, and to hide implementation details from client applications.

Check Compatibility

DS proxy servers connect to remote LDAP directory servers using proxied authorization. The proxied authorization control (OID: `2.16.840.1.113730.3.4.18`) is defined by RFC 4370 *Lightweight Directory Access Protocol (LDAP) Proxied Authorization Control*. If the LDAP directory server does not support proxied authorization, *it cannot be used with DS directory proxy server*.

The following list of LDAP servers *do not support* proxied authorization, and so, do not work with DS directory proxy server:

- Microsoft Active Directory
- Oracle Internet Directory

The following list of LDAP servers support proxied authorization according to their documentation. ForgeRock does not test all servers listed:

- ForgeRock Directory Services
- ApacheDS
- NetIQ eDirectory
- OpenDJ
- OpenLDAP
- Oracle Directory Server Enterprise Edition
- PingDirectory
- Red Hat Directory Server

If your LDAP server does not appear in the lists above, check its documentation regarding support for proxied authorization. Alternatively, check the list of `supportedControl` values on the server's root DSE.

Try DS Directory Proxy

Before installing DS directory proxy server in production, or with a non-DS directory server, try it on your computer.

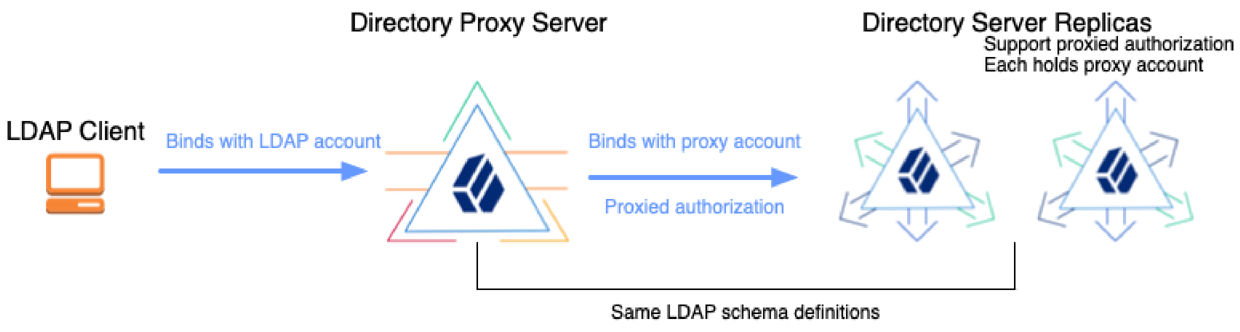

Figure 1. Proxy Configuration

The following examples demonstrate DS directory proxy server forwarding LDAP requests to two DS replicas on your computer. This demonstration includes the following high-level tasks:

- Install two DS directory server replicas as proxied servers.
- Set up the DS directory proxy server to forward requests to the DS directory servers.
- Send LDAP requests to the DS directory proxy server, and observe the results.

The deployment key for installing the server is stored in the environment variable `DEPLOYMENT_KEY`. Install all servers in the same deployment with the same deployment key and deployment key password. For details, read Key Management.

TIP

The DS directory proxy server does not have backup files or directory data to encrypt and decrypt. But it does open secure connections to the remote directory servers, and so must trust the certificates that the remote directory servers present to negotiate TLS.

By default, DS deployments use TLS keys and a CA generated from the deployment key and deployment key password. This is the same deployment key and password used to configure the DS directory servers. Therefore, use the same deployment key and password when configuring DS directory proxy servers, so they can trust the directory server certificates.

Install two DS directory server replicas with the evaluation and proxied server profiles:

```
# Unpack server files:
unzip -q ~/Downloads/DS-7.1.8.zip -d /tmp

# Copy server files before setting up each replica:
mkdir /path/to/opendj && cp -r /tmp/opendj/* /path/to/opendj
mkdir /path/to/replica && cp -r /tmp/opendj/* /path/to/replica

# Set up the servers as replicas of each other
# with StartTLS support for the proxy connections:
/path/to/opendj/setup \
  --deploymentKey $DEPLOYMENT_KEY \
  --deploymentKeyPassword password \
  --hostname localhost \
  --ldapPort 11389 \
  --enableStartTls \
  --ldapsPort 11636 \
  --adminConnectorPort 14444 \
  --rootUserDN uid=admin \
  --rootUserPassword password \
  --profile ds-evaluation \
  --profile ds-proxied-server \
  --set ds-proxied-server/baseDn:dc=example,dc=com \
  --replicationPort 18989 \
  --bootstrapReplicationServer localhost:28989 \
  --acceptLicense \
  --start \
  --quiet

/path/to/replica/setup \
  --deploymentKey $DEPLOYMENT_KEY \
  --deploymentKeyPassword password \
  --hostname localhost \
  --ldapPort 21389 \
  --enableStartTls \
  --ldapsPort 21636 \
  --adminConnectorPort 24444 \
  --rootUserDN uid=admin \
  --rootUserPassword password \
  --profile ds-evaluation \
  --profile ds-proxied-server \
  --set ds-proxied-server/baseDn:dc=example,dc=com \
  --replicationPort 28989 \
  --bootstrapReplicationServer localhost:18989 \
  --acceptLicense \
  --start \
  --quiet

# Update PATH to include DS tools:
export PATH=/path/to/opendj/bin:${PATH}
```

Notice that the examples apply two setup profiles to each replica:

- The DS evaluation setup profile adds sample Example.com data.

  For details, see Install DS for Evaluation.

- The DS proxied server setup profile adds a service account for the proxy server, and sets ACIs to grant the account permission to use proxied authorization. The proxy authenticates to the directory servers with its certificate, whose subject DN is `CN=DS, O=ForgeRock.com`.

  For details, see Install DS For Use With DS Proxy.

Set up a directory proxy server to forward requests to the replicas:

```
# Copy server files before setting up the proxy:
mkdir /path/to/proxy && cp -r /tmp/opendj/* /path/to/proxy

# Set up the proxy server to access the replicas:
/path/to/proxy/setup \
 --serverId proxy \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword password \
 --hostname localhost \
 --ldapPort 1389 \
 --enableStartTls \
 --ldapsPort 1636 \
 --adminConnectorPort 4444 \
 --profile ds-proxy-server \
 --set ds-proxy-
server/bootstrapReplicationServer:"localhost:14444" \
 --set ds-proxy-
server/bootstrapReplicationServer:"localhost:24444" \
 --set ds-proxy-server/rsConnectionSecurity:start-tls \
 --set ds-proxy-server/certNickname:ssl-key-pair \
 --set ds-proxy-server/keyManagerProvider:PKCS12 \
 --set ds-proxy-server/trustManagerProvider:PKCS12 \
 --start \
 --acceptLicense

# Grant access to data through the proxy server:
dsconfig \
 create-global-access-control-policy \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --policy-name "Authenticated access to example.com data" \
 --set authentication-required:true \
 --set request-target-dn-equal-to:"dc=example,dc=com" \
 --set request-target-dn-equal-to:"**,dc=example,dc=com" \
 --set permission:read \
 --set permission:write \
 --set allowed-attribute:"*" \
 --set allowed-attribute:isMemberOf \
 --set allowed-attribute-exception:authPassword \
 --set allowed-attribute-exception:userPassword \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

As you set up only DS servers which all use the same default schema, there is no need to manually align the proxy LDAP schema with the directory server schema.

Send LDAP requests to the DS directory proxy server, and observe the results.

The following example searches the directory through the proxy:

```
$ 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 "ou=people,dc=example,dc=com" \
 "(|(cn=Babs Jensen)(cn=Sam Carter))" \
 cn

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

dn: uid=scarter,ou=People,dc=example,dc=com
cn: Sam Carter
```

The following example modifies directory data through the proxy:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=bjensen,ou=people,dc=example,dc=com \
 --bindPassword hifalutin << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Modified by Babs Jensen
EOF

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

Notice that the bind DNs and passwords are those of the users in the remote directory service.

For more background on each high-level task, read the rest of this page.

Create a Service Account

When preparing to use DS directory proxy servers with directory servers that are not DS servers, create a service account for the proxy to connect to the non-DS 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";)
```

Set Up a Directory Proxy

The deployment key for installing the server is stored in the environment variable `DEPLOYMENT_KEY`. Install all servers in the same deployment with the same deployment key and deployment key password. For details, read Key Management.

Proxy to DS Servers

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Run the **`setup --profile ds-proxy-server`** command.

   The command is located where you installed the files, `/path/to/opendj/setup`:

   The following example sets up a directory proxy server that discovers remote servers based on the DS replication topology. It works with replicas set up using the `ds-proxied-server` setup profile.

   This feature works only with DS servers. If the remote LDAP servers in your deployment are not DS servers, see Proxy to Non-DS Servers.

   This proxy forwards all requests to public naming contexts of remote servers. Generally, this means requests targeting user data, as opposed to the proxy's configuration, schema, or monitoring statistics:

```
$ /path/to/opendj/setup \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword str0ngAdm1nPa55word \
 --hostname ds.example.com \
 --ldapPort 1389 \
 --enableStartTls \
 --ldapsPort 1636 \
 --adminConnectorPort 4444 \
 --profile ds-proxy-server \
 --set ds-proxy-
server/bootstrapReplicationServer:"rs.example.com:4444" \
 --set ds-proxy-server/rsConnectionSecurity:start-tls \
 --set ds-proxy-server/certNickname:ssl-key-pair \
 --set ds-proxy-server/keyManagerProvider:PKCS12 \
 --set ds-proxy-server/trustManagerProvider:PKCS12 \
 --start \
 --acceptLicense
```

   This example uses mutual TLS with a certificate generated with a deployment key and password. Adjust the security settings as required for your deployment.

Proxy to Non-DS Servers

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Run the **`setup --profile ds-proxy-server`** command.

   The command is located where you installed the files, `/path/to/opendj/setup`:

   The following example sets up a directory proxy server that has a static list of remote servers to connect to. It forwards only requests targeting `dc=example,dc=com`:

```
# Initially configure the server with a fake replication
service discovery mechanism:
$ /path/to/opendj/setup \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword str0ngAdm1nPa55word \
 --hostname ds.example.com \
 --ldapPort 1389 \
 --enableStartTls \
 --ldapsPort 1636 \
 --adminConnectorPort 4444 \
 --profile ds-proxy-server \
 --set ds-proxy-server/bootstrapReplicationServer:"fake-
rs.example.com:4444" \
 --set ds-proxy-server/rsConnectionSecurity:start-tls \
 --start \
 --acceptLicense
# Create a static service discovery mechanism:
$ dsconfig \
 create-service-discovery-mechanism \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword str0ngAdm1nPa55word \
 --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 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
# Replace the fake replication service discovery mechanism
with the static one:
$ dsconfig \
 set-backend-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword str0ngAdm1nPa55word \
 --backend-name proxyRoot \
 --set shard:"Static Service Discovery Mechanism" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

This example uses mutual TLS with a certificate generated with a deployment key and password. Adjust the security settings as required for your deployment.

Configure Access Control

1. Explicitly grant appropriate access to remote data.

   The following example grants authenticated users access to data under `dc=example,dc=com`:

   ```
   $ dsconfig \
    create-global-access-control-policy \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword str0ngAdm1nPa55word \
    --policy-name "Authenticated access to example.com data"
   \
   ```

```
  --set authentication-required:true \
  --set request-target-dn-equal-to:"dc=example,dc=com" \
  --set request-target-dn-equal-to:"**,dc=example,dc=com" \
  --set permission:read \
  --set permission:write \
  --set allowed-attribute:"*" \
  --set allowed-attribute:isMemberOf \
  --set allowed-attribute-exception:authPassword \
  --set allowed-attribute-exception:userPassword \
  --usePkcs12TrustStore /path/to/opendj/config/keystore \
  --trustStorePassword:file
 /path/to/opendj/config/keystore.pin \
  --no-prompt
```

DS proxy servers do not use ACIs for access control. Instead, they use global access control policies. By default, the access rights are configured the same as the default settings for a directory server. You no doubt need to adapt these policies for your deployment. For additional examples, see Access Control.

2. Make sure the backend directory servers are properly prepared, as described Create a Service Account.

For more background on LDAP proxy features, see LDAP Proxy.

Default Global Policies

Access control rules are defined using individual access control policy entries. A user's access is defined as the union of all access control rules that apply to that user. In other words, an individual access control rule can only grant additional access and can not remove rights granted by another rule. This approach results in an access control policy which is easier to understand and audit, since all rules can be understood in isolation.

Policy	Settings
Anonymous extended operation and control access	***authentication-required*** • `false` ***allowed-extended-operation*** • `Cancel` • `GetSymmetricKey` • `PasswordModify` • `StartTLS` • `WhoAmI` ***allowed-control*** • `Assertion` • `MatchedValues` • `NoOp` • `PasswordQualityAdvice` • `PermissiveModify` • `PostRead` • `PreRead` • `RealAttrsOnly` • `SimplePagedResults` • `VirtualAttrsOnly` • `AuthorizationIdentity` • `PasswordPolicy` • `TransactionId` • `Vlv`

Policy	Settings
Authenticated extended operation and control access	***authentication-required*** • `true` ***allowed-extended-operation*** • `PasswordPolicyState` ***allowed-control*** • `ManageDsaIt` • `SubEntries` • `RelaxRules` • `SubtreeDelete` • `ServerSideSort`
Schema access	***authentication-required*** • `true` ***request-target-dn-equal-to*** • `cn=schema` ***permission*** • `read` ***allowed-attribute*** • `objectClass` • `@subschema` • `etag` • `ldapSyntaxes` • `modifiersName` • `modifyTimestamp`

Policy	Settings
Root DSE access	***authentication-required*** • `false` ***request-target-dn-equal-to*** • `""` ***permission*** • `read` ***allowed-attribute*** • `objectClass` • `namingContexts` • `subSchemaSubEntry` • `supportedAuthPasswordSchemes` • `supportedControl` • `supportedExtension` • `supportedFeatures` • `supportedLDAPVersion` • `supportedSASLMechanisms` • `supportedTLSCiphers` • `supportedTLSProtocols` • `vendorName` • `vendorVersion` • `fullVendorVersion` • `alive` • `healthy`

Policy	Settings
Monitor access	***authentication-required*** • `true` ***request-target-dn-equal-to*** • `cn=monitor` ***permission*** • `read` ***allowed-attribute*** • `*` • `+`

Align LDAP Schema

Directory servers can reject LDAP change requests that do not comply with LDAP schema. LDAP client applications read LDAP schema definitions from directory servers to determine in advance whether a change request complies with LDAP schema.

When an LDAP client requests LDAP schema from the proxy, the proxy returns *its* LDAP schema. Ideally, the LDAP schema definitions on the proxy match the LDAP schema definitions on the remote directory servers. Otherwise, client applications might check their requests against the proxy's LDAP schema, and yet still see their requests fail with schema violations when the proxy forwards a request to a directory server.

If, after installation, the LDAP schema definitions on the directory servers and the proxy server differ, align the LDAP schema of the proxy server with the LDAP schema of the remote directory servers.

For more information, see LDAP Schema. Schema definitions on a non-DS remote directory server might require translation from another format before you add them on DS directory proxy servers.

Troubleshooting

Common errors with DS directory proxy server installations include the following:

49 (Invalid Credentials)

When LDAP bind requests through the proxy invariably result in an invalid credentials error, but bind requests to the directory server with the same credentials do not, the problem lies with the proxy service account.

The proxy service account must allow bind requests to the directory server. The following example demonstrates a request sent directly to a directory server. The command makes a bind request and then a search request. The directory server is set up according to the instructions in Try DS Directory Proxy:

```
$ 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 "ou=people,dc=example,dc=com" \
 "(|(cn=Babs Jensen)(cn=Sam Carter))" \
 cn

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

dn: uid=scarter,ou=People,dc=example,dc=com
cn: Sam Carter
```

Start with the filtered directory server access log, `logs/filtered-ldap-access.audit.json`, to debug bind failures.

123 (Authorization Denied)

Make sure that access control on the remote LDAP servers lets the proxy service account use the proxied authorization control.

Proxied authorization does not allow operations on remote LDAP servers as the directory superuser (`uid=admin`). Do not connect as directory superuser when trying to access a directory server through the proxy. For administrative requests on remote LDAP servers, access the servers directly. This includes monitoring requests.

It is possible to configure proxied authorization so that an anonymous user (no bind DN, no bind password) can make a request through the proxy server to the remote directory server. ForgeRock recommends that you do not do this, however, as it is less secure.

Many applications perform some operations anonymously, such as reading the root DSE or LDAP schema. These operations are in fact requests to the proxy, not forwarded to remote LDAP servers. For applications to receive an appropriate response for LDAP schema requests, align LDAP schema on the proxy with LDAP schema on the remote LDAP servers as described above.

Install DS For Use With DS Proxy

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Run the **`setup`** command with the `--profile ds-proxied-server` option.

   The example shows the profile used with the evaluation profile. *Add this profile to the list* so proxy servers can access other profiles' data:

   ```
   $ /path/to/opendj/setup \
    --deploymentKey $DEPLOYMENT_KEY \
    --deploymentKeyPassword password \
    --rootUserDN uid=admin \
    --rootUserPassword str0ngAdm1nPa55word \
    --monitorUserPassword str0ngMon1torPa55word \
    --hostname ds.example.com \
    --adminConnectorPort 4444 \
    --ldapPort 1389 \
    --enableStartTls \
    --ldapsPort 1636 \
    --httpsPort 8443 \
    --replicationPort 8989 \
    --bootstrapReplicationServer rs1.example.com:8989 \
    --bootstrapReplicationServer rs2.example.com:8989 \
    --profile ds-evaluation \
    --profile ds-proxied-server \
    --set ds-proxied-server/baseDn:dc=example,dc=com \
    --acceptLicense
   ```

   - The deployment key for installing the server is stored in the environment variable `DEPLOYMENT_KEY`. Install all servers in the same deployment with the same deployment key and deployment key password. For details, read Key Management.
   - The account the DS proxy can use to connect to DS replicas has:
     - Bind DN: The DN from the `--set ds-proxied-server/proxyUserDn` option.

       Default: `uid=proxy`.
     - Certificate subject DN: The DN from the `--set ds-proxied-server/proxyUserCertificateSubjectDn` option.

       Default: `CN=DS, O=ForgeRock.com`.

        - Access to use proxied authorization in the base DNs specified by the multivalued `--set ds-proxied-server/baseDn` option.

          If you do not specify any values for `ds-proxied-server/baseDn`, the proxy user can perform operations with any account as authorization identity. This includes administrator accounts.

          To understand what this means, read Proxied Authorization.

    - The DS proxy server binds using certificate-based authentication with the SASL EXTERNAL mechanism.

      Make sure that the DS replicas' truststores lets them trust the proxy's certificate.

    - The DS proxy server uses proxied authorization to perform operations on the DS replicas.

      The authorization identity for the operations must have appropriate access to the data on the DS replicas.

   For the full list of profiles and parameters, see Default Setup Profiles.

3. Finish configuring the server *before you start it*.

   For a list of optional steps at this stage, see Install DS for Custom Cases.

4. Start the server:

   ```
   $ /path/to/opendj/bin/start-ds
   ```

Install Standalone Servers

Standalone replication servers have no application data backends. They store only changes to directory data. They are dedicated to transmitting replication messages, and to maintaining a replication change log.

Standalone directory servers store replicated copies of application data. These replicas send updates to and receive updates from replication servers. They connect to one replication server at a time, and do not maintain a replication change log.

Each replication server in a deployment connects to all other replication servers. The total number of replication connections, $Total_{conn}$, increases like the square of the number of replication servers. Large deployments that span WAN links can therefore benefit from having fewer replication servers.

$Total_{conn} = (N_{RS} * (N_{RS}-1))/2 + N_{DS}$

Here, N_{RS} is the number of replication servers (standalone or running in a directory server), and N_{DS} is the number of standalone directory servers.

A deployment with only a few standalone replication servers and many standalone directory servers, significantly limits the number of WAN connections for replication:

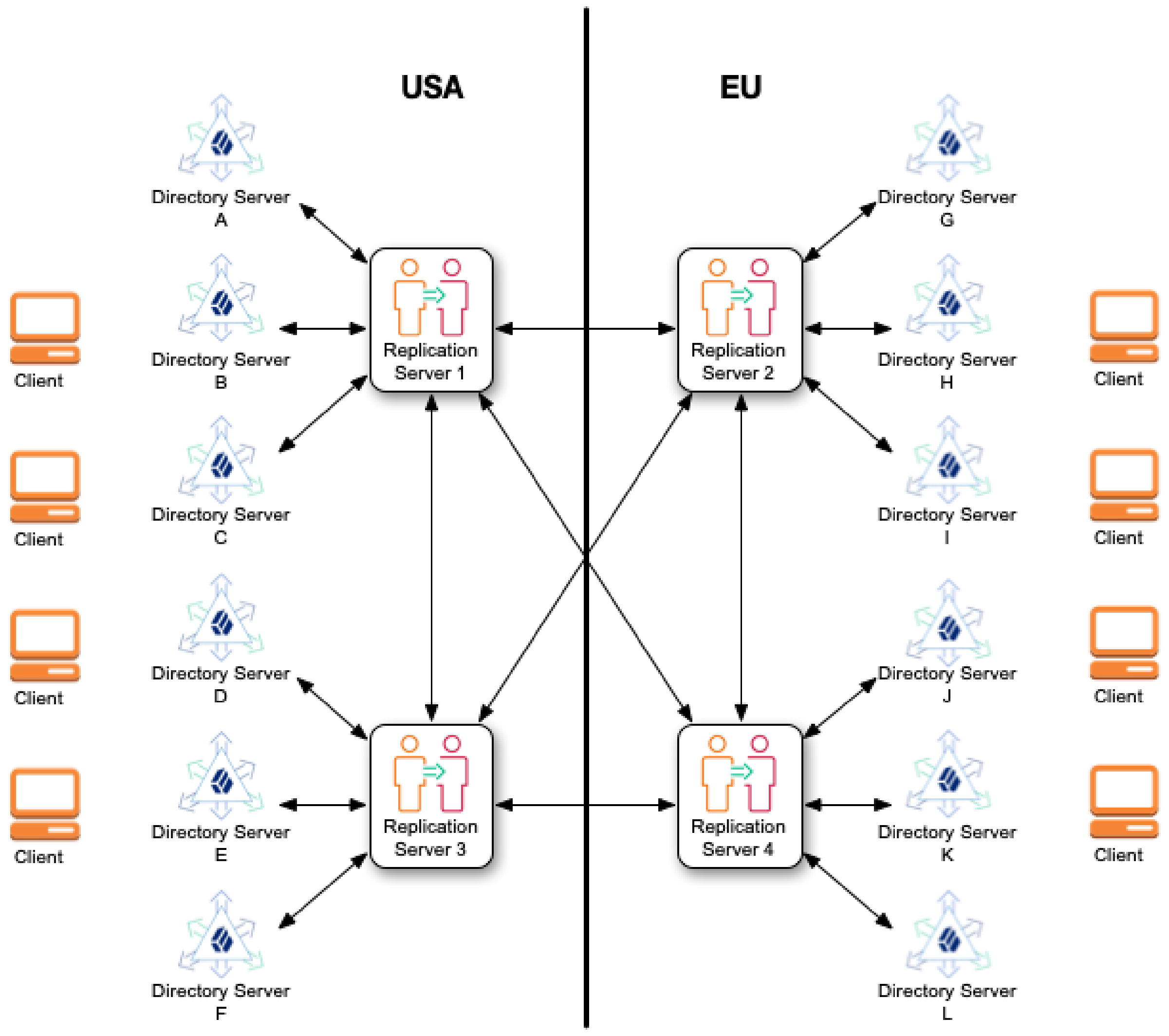

Figure 2. Deployment For Multiple Data Centers

The deployment key for installing the server is stored in the environment variable `DEPLOYMENT_KEY`. Install all servers in the same deployment with the same deployment key and deployment key password. For details, read Key Management.

Set Up Standalone Replication Servers

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Set up a server as a standalone replication server:

```
$ /path/to/opendj/setup \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword password \
 --hostname rs-only.example.com \
 --adminConnectorPort 4444 \
 --replicationPort 8989 \
 --bootstrapReplicationServer rs-only.example.com:8989 \
 --bootstrapReplicationServer rs-only2.example.com:8989 \
 --acceptLicense
```

The standalone replication server has no application data.

It does have LDAP schema and changelog data. If you plan to add any additional schema to the replicas as part of the setup process, also add the schema to this server before starting it.

3. Start the server:

```
$ /path/to/opendj/bin/start-ds
```

4. Repeat the previous steps on additional systems until you have sufficient replication servers to meet your availability requirements.

   To ensure availability, add at least one additional replication server per location. The following example adds a second standalone replication server:

```
$ /path/to/opendj/setup \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDN uid=admin \
 --rootUserPassword password \
 --hostname rs-only2.example.com \
 --adminConnectorPort 4444 \
 --replicationPort 8989 \
 --bootstrapReplicationServer rs-only.example.com:8989 \
 --bootstrapReplicationServer rs-only2.example.com:8989 \
 --acceptLicense
```

   The standalone replication servers use each other as bootstrap servers to discover other servers in the deployment.

Set Up Standalone Directory Servers

1. Before proceeding, install the server files.
   For details, see Unpack Files.
2. Set up the server as a directory server.

   Notice that the `--bootstrapReplicationServer` references the replication servers set up according to the steps in Set Up Standalone Replication Servers.

   The `--replicationPort` option is not used, because this is a standalone directory server:

   ```
   $ /path/to/opendj/setup \
    --serverId evaluation-only \
    --deploymentKey $DEPLOYMENT_KEY \
    --deploymentKeyPassword password \
    --rootUserDN uid=admin \
    --rootUserPassword password \
    --adminConnectorPort 4444 \
    --hostname ds-only.example.com \
    --ldapPort 1389 \
    --enableStartTls \
    --ldapsPort 1636 \
    --httpsPort 8443 \
    --bootstrapReplicationServer rs-only.example.com:8989 \
    --bootstrapReplicationServer rs-only2.example.com:8989 \
    --profile ds-evaluation \
    --acceptLicense
   ```

3. Finish configuring the server *before you start it*.

   For a list of optional steps at this stage, see Install DS for Custom Cases.

4. Start the server:

   ```
   $ /path/to/opendj/bin/start-ds
   ```

5. Repeat the previous steps on additional systems until you have sufficient directory servers to meet your availability and performance requirements.

   To ensure availability, add at least one additional directory server per location.

Install With Existing Cryptographic Keys

NOTE

When you set up a DS server with your own keys for PKI, you must still use a deployment key and password.

In this case, the deployment key and password generate the shared master key used to protect secret keys for encrypted data, such as backup files. If you have not yet learned about deployment keys, read Deployment Keys.

The **`setup`** command has options to simplify setting up a server with existing keys:

For…	Use…

Keystores containing server key pairs	`--useJavaKeyStore`
	`--useJceKeyStore`
	`--usePkcs11KeyStore`
	`--usePkcs12KeyStore`
	`-W, --keyStorePassword[:env|:file]`
	`-N, --certNickname`

Truststores containing trusted CA or self-signed certificates	`--useJavaTrustStore`
	`--useJceTrustStore`
	`--usePkcs11TrustStore`
	`--usePkcs12TrustStore`
	`-T, --trustStorePassword[:env|:file]`

Important features to be aware of:

- If the keystore file that holds the server key pair protects the server key with a password, that password must match the password for the entire store.

  DS does not support separate keystore and key passwords in keystore files.
- If you are using an HSM, also see PKCS#11 Hardware Security Module.
- If you are using PEM format keys, see PEM Format Keys instead.

Follow steps similar to these to install a DS replica with existing cryptographic keys:

1. Before proceeding, install the server files.

   For details, see Unpack Files.
2. Run the **`setup`** command with the appropriate options.

   The following example uses a PKCS#12 keystore file with the server's key pair, and a PKCS#12 truststore file with the CA's certificate.

   This example installs the server with the evaluation setup profile. Adapt the command for your use:

```
# Set up a directory server for evaluation using existing
keys:
$ /path/to/opendj/setup \
 --serverId evaluation-only \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --usePkcs12TrustStore /path/to/truststore \
 --trustStorePassword password \
 --certNickname ssl-key-pair \
 --usePkcs12KeyStore /path/to/keystore \
 --keyStorePassword password \
 --rootUserDN uid=admin \
 --rootUserPassword password \
 --monitorUserPassword password \
 --hostname localhost \
 --adminConnectorPort 4444 \
 --ldapPort 1389 \
 --enableStartTls \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --replicationPort 8989 \
 --bootstrapReplicationServer localhost:8989 \
 --profile ds-evaluation \
 --acceptLicense
```

3. Finish configuring the server.

4. Start the server:

```
$ /path/to/opendj/bin/start-ds
```

When you set up the server to use existing keystore files, the server configuration directly references those files. If you read the server configuration, you find that a Key Manager Provider references the keystore, and that a Trust Manager Provider references the truststore.

If you provide keystore and truststore passwords as strings, the **`setup`** command records them in files in the `opendj/config` directory.

For details on using variables instead, see Property Value Substitution.

Install a REST to LDAP Gateway

A REST to LDAP gateway web application translates an HTTP request into one or more LDAP requests. The translation depends on the specific REST to LDAP gateway configuration.

An identity mapper translates the user identity into an LDAP identity for the bind. For background information, see Identity Mappers.

Then the REST to LDAP mapping defines how the REST JSON resource corresponds to LDAP entries. The gateway handles the mapping configuration in the same way as an HTTP connection handler.

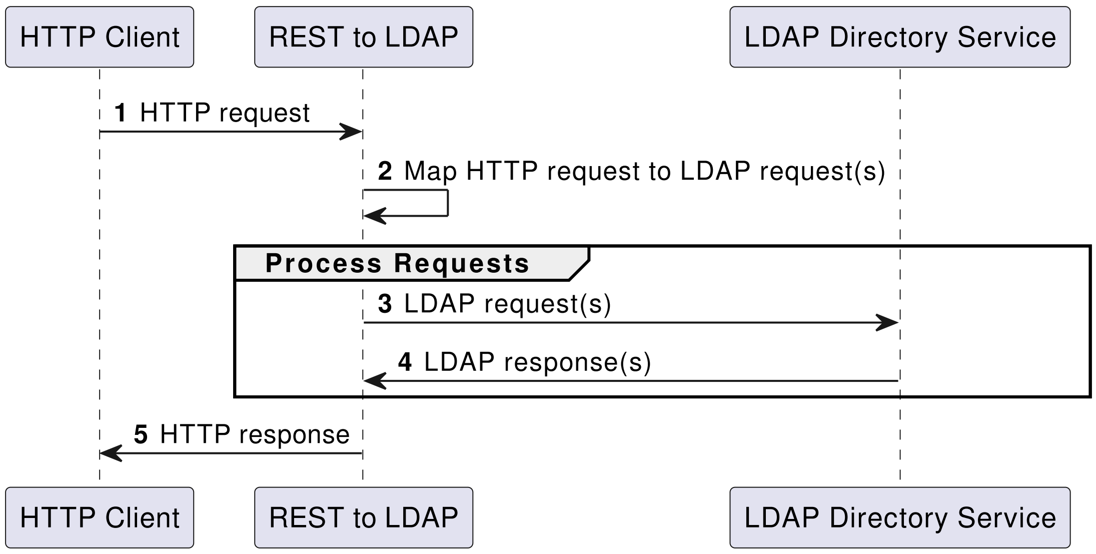

Figure 3. Requests Through a REST to LDAP Gateway

The REST to LDAP gateway functions as a web application in a web application container. The REST to LDAP gateway runs independently of the LDAPv3 directory service. As an alternative to the gateway, you can configure HTTP access to a directory server, as described in Configure HTTP User APIs.

You configure the gateway to access your directory service by editing configuration files in the deployed web application:

`WEB-INF/classes/config.json`

This file defines how the gateway connects to LDAP directory servers, and how user identities extracted from HTTP requests map to LDAP user identities.

For details, see Gateway LDAP Connections.

`WEB-INF/classes/logging.properties`

This file defines logging properties, and can be used when the gateway runs in Apache Tomcat.

`WEB-INF/classes/rest2ldap/rest2ldap.json`

This file defines which LDAP features the gateway uses.

For details, see Gateway LDAP Features.

`WEB-INF/classes/rest2ldap/endpoints/api/example-v1.json`
This file defines JSON resource to LDAP entry mappings.

You can edit this file, and define additional files for alternative APIs and versions of APIs. For details, see API Configuration.

Follow these steps to install the REST to LDAP gateway:

1. Prepare for installation.
2. Deploy the .war file according to the instructions for your web application container.

   If you are using Wildfly, you must unzip the .war file into the deployment directory.
3. Edit the configuration files in the deployed gateway web application.

   At minimum adjust the following configuration settings in `WEB-INF/classes/config.json`:

   - `primaryLDAPServers`: Set the correct directory server hostnames and port numbers.
   - `authentication`: Set the correct simple bind credentials.

     The LDAP account used to authenticate needs to perform proxied authorization, as described in Proxied Authorization.

     The default sample configuration works with generated example data, and with the sample data imported when you set up the directory server for evaluation, as shown in Install DS for Evaluation. If your data is different, then you must also change the JSON resource to LDAP entry mapping settings, as described in API Configuration.

     For details regarding the configuration, see REST to LDAP Reference.

     When connecting to a directory service over LDAPS or LDAP and StartTLS, you can configure the trust manager to use a file-based truststore for server certificates that the gateway should trust. This allows the gateway to validate server certificates signed, for example, by a certificate authority that is not recognized by the Java environment when setting up LDAPS or StartTLS connections.

     See Key Management for an example of how to use the Java **`keytool`** command to import a server certificate into a truststore file.
4. If necessary, adjust the log level.

   Log levels are defined in java.util.logging.Level.

By default, the log level is set to `INFO`, and the gateway logs HTTP request-related messages. To have the gateway log LDAP request-related messages, set the log level to `FINEST` in one of the following ways:

a. If the REST to LDAP gateway runs in Apache Tomcat, edit `WEB-INF/classes/logging.properties` to set `org.forgerock.opendj.rest2ldap.level = FINEST`. For details on Tomcat's implementation of the logging API, see Logging in Tomcat.

   Messages are written to `CATALINA_BASE/logs/rest2ldap.yyyy-MM-dd.log`.

b. If the REST to LDAP gateway runs in a different container, set the log level as described in the documentation.

   Messages are written to the container log.

5. Restart the REST to LDAP gateway or the web application container to make sure the configuration changes are taken into account.

6. Make sure that the directory service is up, and then check that the gateway is connecting correctly.

   The following command reads Babs Jensen's entry through the gateway to a directory server set up for evaluation, as shown in Install DS for Evaluation. In this example, the gateway is deployed under `/rest2ldap`:

```
$ curl \
 --user bjensen:hifalutin \
 --cacert ca-cert.pem \
 https://localhost:8443/rest2ldap/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"
  }
}
```

If you generated example data, Babs Jensen's entry is not included. Instead, try a generated user such as `https://user.0:password@localhost:8443/rest2ldap/api/users/user.0`.

7. Configure your web application container to use HTTPS for secure connections to the gateway.

   See your web application container documentation for details.

Install a DSML Gateway

The DSML gateway web application translates each HTTP request into one or more LDAP requests. The translation depends on the DSML protocol. For authentication, you must configure how HTTP user IDs map to LDAP identities.

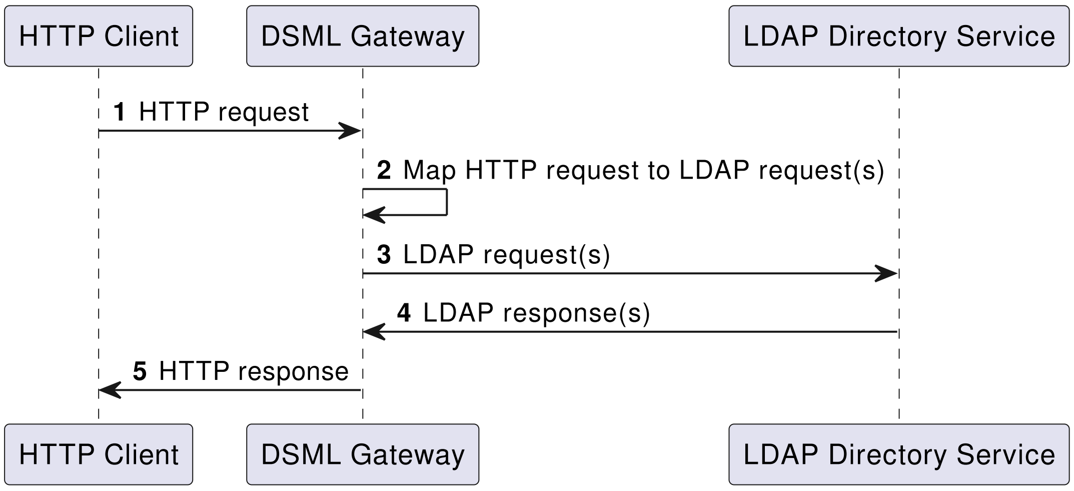

Figure 4. Requests Through a DSML Gateway

The DSML gateway functions as a web application in a web application container.

The DSML gateway runs independently of the directory service.

You configure the gateway to access a directory service by editing parameters in the gateway configuration file, `WEB-INF/web.xml`:

1. Prepare for installation.
2. Deploy the .war file according to the instructions for your web application container.
3. Edit `WEB-INF/web.xml` to ensure the parameters are correct.

   For details, see Configure DSML Access.
4. Configure your web application container to use HTTPS for secure connections to the gateway.

   See your web application container documentation for details.
5. Restart the web application according to the instructions for your web application container.

Configure DSML Access

Directory Services Markup Language (DSML) client access is implemented as a servlet web application. You edit the `WEB-INF/web.xml` file after deploying the web application.

The list of DSML configuration parameters are the following:

`ldap.host`
: The hostname of the underlying directory service.

  Default: `localhost`

`ldap.port`
: The LDAP port number of the underlying directory service.

  Default: `389`

`ldap.userdn`
: Optional parameter specifying the DN to bind to the underlying directory service.

  Default: anonymous bind

`ldap.userpassword`
: Optional parameter specifying the password to bind to the underlying directory service.

  Default: anonymous bind

`ldap.authzidtypeisid`
: Use this parameter to set up the DSML gateway to do HTTP Basic Access Authentication, given the appropriate mapping between the user ID, and the user’s entry in the directory.

This takes a boolean parameter specifying whether the HTTP Authorization header field’s Basic credentials in the request hold a plain ID, rather than a DN.

If set to `true`, the gateway performs an LDAP SASL bind using SASL plain, enabled by default in DS servers to look for an exact match between a `uid` in the server, and the plain ID from the header.

In other words, if the plain ID is `bjensen`, then the bind DN is `uid=bjensen,ou=people,dc=example,dc=com`.

Configure DS identity mappers as necessary to use a different attribute than `uid`. For background information, see Identity Mappers.

Default: `false`

`ldap.usessl`

Whether `ldap.port` uses LDAPS.

Default: `false`

`ldap.usestarttls`

Whether to use StartTLS when connecting to `ldap.port`.

Default: `false`

`ldap.trustall`

Whether to blindly trust all server certificates when using LDAPS or StartTLS.

Default: `false`

`ldap.truststore.path`

The truststore used to verify server certificates when using LDAPS or StartTLS.

Required when using LDAPS or StartTLS and `ldap.trustall is false`.

`ldap.truststore.password`

The password to read the truststore.

Required when using a truststore with a password.

For initial testing purposes, try JXplorer, where the DSML Service is: `/`*`webapp-dir`*`/DSMLServlet`. The *`webapp-dir`* refers to the name of the directory holding the DSML `.war`.

Uninstallation

Uninstall .Zip

Follow these steps to remove software installed from the cross-platform .zip:

1. Log in as the user who installed and runs the server.
2. Stop the server:

   ```
   $ /path/to/opendj/bin/stop-ds --quiet
   ```

3. Delete the files manually:

   ```
   $ rm -rf /path/to/opendj
   ```

Uninstall the Debian Package

When you uninstall the Debian package from the command-line, the server is stopped if it is running:

1. Purge the package from your system:

   ```
   $ sudo dpkg --purge opendj
   ```

2. Remove any remaining server configuration files and directory data:

   ```
   $ sudo rm -rf /opt/opendj
   ```

Uninstall the RPM Package

When you uninstall the RPM package from the command-line, the server is stopped if it is running:

1. Remove the package from your system:

   ```
   # rpm -e opendj
   ```

2. Remove the server configuration files and any directory data:

   ```
   $ sudo rm -rf /opt/opendj
   ```

Uninstall the Windows MSI

When you uninstall the files installed from the Windows installer package, only the installed files are removed:

1. Remove installed files in one of the following ways:
    a. Use Windows Control Panel.
        - Open Windows Control Panel and browse to the page to uninstall a program.
        - Find the ForgeRock directory service in the list and uninstall it.

    b. Use the **`msiexec`** command.

    The following command quietly removes installed files:

    ```
    C:\> msiexec /x DS-7.1.8.msi /q
    ```
2. Manually remove the server configuration files and any directory data.

File Layout

DS software installs and creates the following files and directories. The following table is not meant to be exhaustive.

File or Directory	Description
`bak`	Directory intended for local backup data.
`bat`	Windows command-line tools.
`bin`	UNIX/Linux command-line tools.
`changelogDb`	Backend for replication changelog data.
`classes`	Directory added to the server classpath, permitting individual classes to be patched.
`config`	(Optionally) immutable server configuration files.
`config/audit-handlers`	Templates for configuring external Common Audit event handlers.
`config/config.ldif`	LDIF representation of current DS server configuration.

File or Directory	Description
`config/keystore` `config/keystore.pin`	Keystore and password (`.pin`) files for servers using PKI based on deployment keys.
`config/MakeLDIF`	Templates for use with the **`makeldif`** LDIF generation tool.
`db`	Default directory for backend database files. For details, see Data Storage.
`extlib`	Directory for additional .jar files used by your custom plugins. If the instance path is not the same as the binaries, copy additional files into the `instance-path/extlib/` directory.
`import-tmp`	Working directory used when importing LDIF data.
`ldif`	Directory for saving LDIF export files.
`legal-notices`	License information.
`lib`	Scripts and libraries shipped with DS servers.
`lib/extensions`	Directory for custom plugins.
`locks`	Lock files that prevent more than one process from using the same backend.
`logs`	Access, errors, audit, and replication logs.
`logs/server.pid`	Contains the process ID for a running server.
`opendj_logo.png`	DS splash logo.
`README`	About DS servers.
`samples` (1)	Samples for use with DS servers, such as: • A sample `Dockerfile` and related files for building custom DS Docker images. • A sample Grafana dashboard demonstrating how to graph DS server metrics stored in a Prometheus database. • Sample server plugins and extensions.

File or Directory	Description
`setup`	UNIX/Linux setup tool.
`setup.bat`	Windows setup tool.
`snmp`	SNMP support files.
`template`	Templates for setting up a server instance.
`template/setup-profiles`	Profile scripts to configure directory servers for specific use cases.
`upgrade`	UNIX/Linux upgrade tool.
`upgrade.bat`	Windows upgrade tool.
`var`	Files the DS server writes to during operation. Do not modify or move files in the `var` directory.
`var/archived-configs`	Snapshots of the main server configuration file, `config/config.ldif`. The server writes a compressed snapshot file when the configuration is changed.
`var/config.ldif.startok`	The most recent version of the main server configuration file that the server successfully started with.

[1] The samples are provided on an "as is" basis. ForgeRock does not guarantee the individual success developers may have in implementing the samples on their development platforms or in production configurations.

For details about how to try the samples, see the accompanying `README.md` files.

Was this helpful?