Maintenance

This guide covers recurring administrative operations.

Tools

Run DS command-line tools.

Server Process

Start, stop, restart DS.

Backup/Restore

Backup and restore data.

Resource Limits

Set limits for user and application.

Tuning

Tune server performance.

Troubleshooting

Solve common problems.

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.

Maintenance Tools

Server Commands

- Add DS server command-line tools to your PATH:
    1. Bash
    2. PowerShell

```
$ export PATH=/path/to/opendj/bin:${PATH}
```

```
PS C:\path\to> $env:PATH += ";C:\path\to\opendj\bat"
```

- For reference information, use the `--help` option with any DS tool.
- All commands call Java programs. This means every command starts a JVM, so it takes longer to start than a native binary.

DS running on…	DS installed from…	Default path to tools…
Linux distributions	.zip	`/path/to/opendj/bin`
Linux distributions	.deb, .rpm	`/opt/opendj/bin`
Microsoft Windows	.zip	`C:\path\to\opendj\bat`

The installation and upgrade tools, **`setup`**, and **`upgrade`**, are found in the parent directory of the other tools. These tools are not used for everyday administration.

Commands	Constraints
`dsbackup` **`dsconfig`** **`export-ldif`** **`import-ldif`** **`rebuild-index`** **`setup`** **`setup-profile`** **`start-ds`**	When the server is offline, or when running commands in offline mode, these commands can modify server files. They must, therefore, access server files as a user who has the same filesystem permissions as the user who installs and runs the server. For most systems, the simplest way to achieve this is to run the command as the same user who installs and runs the server. When following best practices for auditing and separation of duty, provision administrative and server user accounts with compatible group or access control list permissions.

Commands	Constraints
backendstat **create-rc-script** **encode-password** **setup** **setup-profile** **start-ds** **supportextract** **upgrade** **windows-service**	These commands must be used with the local DS server in the same installation as the tools. These commands are not useful with non-DS servers.
dsbackup **changelogstat** **dsconfig** **dsrepl** **encode-password** **export-ldif** **import-ldif** **manage-account** **manage-tasks** **rebuild-index** **status** **stop-ds** **verify-index**	These commands must be used with DS servers having the same version as the command. These commands are not useful with non-DS servers.
makeldif	This command depends on template files. The template files can make use of configuration files installed with DS servers under `config/MakeLDIF/`. The LDIF output can be used with any directory server.
base64 **ldapcompare** **ldapdelete** **ldapmodify** **ldappasswordmodify** **ldapsearch** **ldifdiff** **ldifmodify** **ldifsearch**	These commands can be used independently of DS servers, and are not tied to a specific version.

Command[1]	Description
addrate	Measure add and delete throughput and response time.
authrate	Measure bind throughput and response time.
backendstat	Debug databases for pluggable backends.
base64	Encode and decode data in base64 format. Base64-encoding represents binary data in ASCII, and can be used to encode character strings in LDIF, for example.
changelogstat	Debug file-based changelog databases.
create-rc-script (UNIX)	Generate a script you can use to start, stop, and restart the server, either directly, or at system boot and shutdown. Use **create-rc-script -f** *script-file*. This lets you register and manage DS servers as services on UNIX and Linux systems.
dsbackup	Back up or restore directory data.
dskeymgr	Generate a deployment key, a shared master key, a private CA certificate based on a deployment key and password, or a key pair with the certificate signed by the private CA.

Command[1]	Description
dsconfig	The **dsconfig** command is the primary command-line tool for viewing and editing DS server configurations. When started without arguments, **dsconfig** prompts you for administration connection information. Once connected to a running server, it presents you with a menu-driven interface to the server configuration. To edit the configuration when the server is not running, use the `--offline` command. Some advanced properties are not visible by default when you run the **dsconfig** command interactively. Use the `--advanced` option to access advanced properties. When you pass connection information, subcommands, and additional options to **dsconfig** , the command runs in script mode, so it is not interactive. You can prepare **dsconfig** batch scripts with the `--commandFilePath` option in interactive mode, then read from the batch file with the `--batchFilePath` option in script mode. Batch files can be useful when you have many **dsconfig** commands to run, and want to avoid starting the JVM for each command. Alternatively, you can read commands from standard input with the `--batch` option.
dsrepl	Manage data replication between directory servers to keep their contents in sync.
encode-password	Encode a plaintext password according to one of the available storage schemes.
export-ldif	Export directory data to LDIF, the standard, portable, text-based representation of directory content.
import-ldif	Load LDIF content into the directory, which overwrites existing data. It cannot be used to append data to the backend database.
ldapcompare	Compare the attribute values you specify with those stored on entries in the directory.
ldapdelete	Delete one entry or an entire branch of subordinate entries in the directory.

Command[1]	Description
ldapmodify	Modify the specified attribute values for the specified entries.
ldappasswordmodify	Modify user passwords.
ldapsearch	Search a branch of directory data for entries that match the LDAP filter you specify.
ldifdiff	Display differences between two LDIF files. The output is LDIF.
ldifmodify	Similar to the **ldapmodify** command, modify specified attribute values for specified entries in an LDIF file.
ldifsearch	Similar to the **ldapsearch** command, search a branch of data in LDIF for entries matching the LDAP filter you specify.
makeldif	Generate directory data in LDIF based on templates that define how the data should appear. The **makeldif** command generates test data that mimics data expected in production, and does not compromise real, potentially private information.
manage-account	Lock and unlock user accounts, and view and manipulate password policy state information.
manage-tasks	View information about tasks scheduled to run in the server, and cancel specified tasks.
modrate	Measure modification throughput and response time.
rebuild-index	Rebuild an index stored in an indexed backend.
searchrate	Measure search throughput and response time.
setup-profile	Configure a setup profile after initial installation.
start-ds	Start one DS server.
status	Display information about the server.
stop-ds	Stop one DS server.
supportextract	Collect troubleshooting information for technical support purposes.

Command[1]	Description
`verify-index`	Verify that an index stored in an indexed backend is not corrupt.
`windows-service` (Windows)	Register and manage one DS server as a Windows service.

[1] UNIX names for the commands. Equivalent Windows commands have .bat extensions.

Trusted Certificates

When a client tool initiates a secure connection to a server, the server presents its digital certificate.

The tool must determine whether it trusts the server certificate and continues to negotiate a secure connection, or does not trust the server certificate and drops the connection. To trust the server certificate, the tool's truststore must contain the trusted certificate. The trusted certificate is a CA certificate, or the self-signed server certificate.

The following table explains how the tools locate the truststore.

Truststore Option	Truststore Used
None	The default truststore, `user.home/.opendj/keystore`, where *user.home* is the Java system property. *user.home* is `$HOME` on Linux and UNIX, and `%USERPROFILE%` on Windows. The keystore password is `OpenDJ`. Neither the file name, nor the password can be changed. • In interactive mode, DS command-line tools prompt for approval to trust an unrecognized certificate, and whether to store it in the default truststore for future use. • In silent mode, the tools rely on the default truststore.
`--use<Type>TrustStore {trustStorePath}`	Only the specified truststore is used. The <*Type*> in the option name reflects the trust store type. The tool fails with an error if it cannot trust the server certificate.

Default Settings

You can set defaults in the `~/.opendj/tools.properties` file, as in the following example:

```
hostname=localhost
port=4444
bindDN=uid=admin
useSsl=true
trustAll=true
```

The file location on Windows is `%UserProfile%\.opendj\tools.properties`.

Server Processes

Start a Server

- Start the server in the background:

  ```
  $ start-ds
  ```

  Alternatively, specify the `--no-detach` option to start the server in the foreground.

- (Linux) If the DS server was installed from a .deb or .rpm package, then service management scripts were created at setup time:

  ```
  centos# service opendj start

  Starting opendj (via systemctl):                           [
  OK  ]
  ```

  ```
  ubuntu$ sudo service opendj start

  $Starting opendj: > SUCCESS.
  ```

- (UNIX) Create an RC script, and use the script to start the server.

  Unless you run DS servers on Linux as root, use the `--userName` *`userName`* option to specify the user who installed the server:

  ```
  $ sudo create-rc-script --outputFile /etc/init.d/opendj --
  userName opendj
  ```

```
$ sudo /etc/init.d/opendj start
```

For example, if you run the DS server on Linux as root, you can use the RC script to start the server at system boot, and to stop the server at system shutdown:

```
$ sudo update-rc.d opendj defaults

update-rc.d: warning: /etc/init.d/opendj missing LSB
information
update-rc.d: see <http://wiki.debian.org/LSBInitScripts>
 Adding system startup for /etc/init.d/opendj ...
   /etc/rc0.d/K20opendj -> ../init.d/opendj
   /etc/rc1.d/K20opendj -> ../init.d/opendj
   /etc/rc6.d/K20opendj -> ../init.d/opendj
   /etc/rc2.d/S20opendj -> ../init.d/opendj
   /etc/rc3.d/S20opendj -> ../init.d/opendj
   /etc/rc4.d/S20opendj -> ../init.d/opendj
   /etc/rc5.d/S20opendj -> ../init.d/opendj
```

Alternatively, generate a service file with the `--systemdService` option, and use `systemd` to manage the service.

- (Windows) Register the DS server as a Windows service:

```
C:\path\to\opendj\bat> windows-service.bat --enableService
```

Manage the service with Windows-native administration tools.

Stop a Server

Although DS servers are designed to recover from failure and disorderly shutdown, it is safer to shut the server down cleanly, because a clean shutdown reduces startup delays. During startup, the server attempts to recover database backend state. Clean shutdown prevents situations where the server cannot recover automatically.

Clean Server Retirement

1. Before shutting down the system where the server is running, and before detaching any storage used for directory data, cleanly stop the server using one of the following techniques:
   - Use the **`stop-ds`** command:

```
$ stop-ds
```

- (Linux) If the DS server was installed from a .deb or .rpm package, then service management scripts were created at setup time:

```
centos# service opendj stop

Stopping opendj (via systemctl):
[  OK  ]
```

```
ubuntu$ sudo service opendj stop

$Stopping opendj: ... > SUCCESS.
```

- (UNIX) Create an RC script, and then use the script to stop the server:

```
$ sudo create-rc-script --outputFile /etc/init.d/opendj
--userName opendj

$ sudo /etc/init.d/opendj stop
```

- (Windows) Register the DS server once as a Windows service:

```
C:\path\to\opendj\bat> windows-service.bat --
enableService
```

  Manage the service with Windows-native administration tools.

Do not intentionally kill the DS server process unless the server is completely unresponsive.

+ When stopping cleanly, the server writes state information to database backends, and releases locks that it holds on database files.

Restart a Server

- Use the **`stop-ds`** command:

```
$ stop-ds --restart
```

- (Linux) If the DS server was installed from a .deb or .rpm package, then service management scripts were created at setup time:

```
centos# service opendj restart

Restarting opendj (via systemctl):                         [
OK  ]
```

```
ubuntu$ sudo service opendj restart

$Stopping opendj: ... > SUCCESS.

$Starting opendj: > SUCCESS.
```

- (UNIX) Create an RC script, and then use the script to stop the server:

  ```
  $ sudo create-rc-script --outputFile /etc/init.d/opendj --
  userName opendj

  $ sudo /etc/init.d/opendj restart
  ```

- (Windows) Register the DS server once as a Windows service:

  ```
  C:\path\to\opendj\bat> windows-service.bat --enableService
  ```

  Manage the service with Windows-native administration tools.

Server Tasks

The following server administration commands can be run in online and offline mode. They invoke data-intensive operations, and so potentially take a long time to complete. The links below are to the reference documentation for each command:

- dsbackup
- export-ldif
- import-ldif
- rebuild-index

When you run these commands in online mode, they run as *tasks* on the server. Server tasks are scheduled operations that can run one or more times as long as the server is up. For example, you can schedule the **`dsbackup`** and **`export-ldif`** commands to run recurrently in order to back up server data on a regular basis.

You schedule a task as a directory administrator, sending the request to the administration port. You can therefore schedule a task on a remote server if you choose. When you schedule a task on a server, the command returns immediately, yet the task

can start later, and might run for a long time before it completes. You can access tasks by using the manage-tasks command.

Although you can schedule a server task on a remote server, *the data for the task must be accessible to the server locally*. For example, when you schedule a backup task on a remote server, that server writes backup files to a file system on the remote server. Similarly, when you schedule a restore task on a remote server, that server restores backup files from a file system on the remote server.

The reference documentation describes the available options for each command:

- Configure email notification for success and failure
- Define alternatives on failure
- Start tasks immediately (`--start 0`)
- Schedule tasks to start at any time in the future

Server Recovery

DS servers can restart after a crash or after the server process is killed abruptly. After disorderly shutdown, the DS server must recover its database backends. Generally, DS servers return to service quickly.

Database recovery messages are found in the database log file, such as `/path/to/opendj/db/userData/dj.log` .

The following example shows two example messages from the recovery log. The first message is written at the beginning of the recovery process. The second message is written at the end of the process:

```
[/path/to/opendj/db/userData]Recovery underway, found end of log
...
[/path/to/opendj/db/userData]Recovery finished: Recovery Info ...
```

The JVM's heap-based database cache is lost when the server stops or crashes. The cache must therefore be reconstructed from the directory database files. Database files might still be in the filesystem cache on restart, but rebuilding the JVM's heap-based database cache takes time. DS servers start accepting client requests before this process is complete.

Backup and Restore

IMPORTANT

IMPORTANT

- Backup archives are *not guaranteed to be compatible* across major and minor server releases. *Restore backups only on directory servers of the same major or minor version.*

  To share data between servers of different versions, either use replication, or use LDIF.

- DS servers use cryptographic keys to sign and verify the integrity of backup files, and to encrypt data. Servers protect these keys by encrypting them with the shared master key for a deployment. For portability, servers store the encrypted keys in the backup files.

  Any server can therefore restore a backup taken with the same server version, *as long as it holds a copy of the shared master key used to encrypt the keys*.

Back Up

When you set up a directory server, the process creates a `/path/to/opendj/bak/` directory. You can use this for backups if you have enough local disk space, and when developing or testing backup processes. In deployment, store backups remotely to avoid losing your data and backups in the same crash.

Back Up Data (Server Task)

When you schedule a backup as a server task, the DS server manages task completion. The server must be running when you schedule the task, and when the task runs:

1. Schedule the task on a running server, binding as a user with the `backend-backup` administrative privilege.

   The following example schedules an immediate backup task for the `dsEvaluation` backend:

   ```
   $ dsbackup \
    create \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePassword:file
   /path/to/opendj/config/keystore.pin \
   ```

```
  --backupLocation bak \
  --backendName dsEvaluation
```

To back up all backends, omit the `--backendName` option.

To back up more than one backend, specify the `--backendName` option multiple times.

For details, see dsbackup.

Back Up Data (Scheduled Task)

When you schedule a backup as a server task, the DS server manages task completion. The server must be running when you schedule the task, and when the task runs:

1. Schedule backups using the **`crontab`** format with the `--recurringTask` option.

   The following example schedules nightly online backup of all user data at 2 AM, notifying `diradmin@example.com` when finished, or on error:

   ```
   $ dsbackup \
    create \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePassword:file
   /path/to/opendj/config/keystore.pin \
    --backupLocation bak \
    --recurringTask "00 02 * * *" \
    --description "Nightly backup at 2 AM" \
    --taskId NightlyBackup \
    --completionNotify diradmin@example.com \
    --errorNotify diradmin@example.com
   ```

   For details, see dsbackup.

Back Up Data (External Command)

When you back up data without contacting the server, the **`dsbackup create`** command runs as an external command, independent of the server process. It backs up the data whether the server is running or not.

NOTE

When you back up LDIF-based backends with this method, the command does not lock the files. To avoid corrupting the backup files, do not run the **`dsbackup create --offline`** command on an LDIF backend simultaneously with any changes to the backend.

This applies to LDIF backends, schema files, and the task backend, for example.

Use this method to schedule backup with a third-party tool, such as the **`cron`** command:

1. Back up data without contacting the server process, and use the `--offline` option.

   The following example backs up the `dsEvaluation` backend immediately:

   ```
   $ dsbackup \
    create \
    --offline \
    --backupLocation bak \
    --backendName dsEvaluation
   ```

   To back up all backends, omit the `--backendName` option.

   To back up more than one backend, specify the `--backendName` option multiple times.

   For details, see dsbackup.

Back Up Configuration Files

When you back up directory data using the **`dsbackup`** command, you do not back up server configuration files. The server stores configuration files under the `/path/to/opendj/config/` directory.

The server records snapshots of its configuration under the `/path/to/opendj/var/` directory. You can use snapshots to recover from misconfiguration performed with the **`dsconfig`** command. *Snapshots only reflect the main configuration file, `config.ldif`.*

1. Stop the server:

```
$ stop-ds
```

2. Back up the configuration files:

```
$ tar -zcvf backup-config-$(date +%s).tar.gz config
```

   By default, this backup includes the server keystore, so store it securely.

3. Start the server:

```
$ start-ds
```

Back Up Using Snapshots

ForgeRock recommends using the **dsbackup** command when possible for backup and restore operations. You can use snapshot technology as an alternative to the **dsbackup** command, but you must be careful how you use it.

While DS directory servers are running, database backend cleanup operations write data even when there are no pending client or replication operations. An ongoing file system backup operation may record database log files that are not in sync with each other.

Successful recovery after restore is only guaranteed under certain conditions.

The snapshots must:

- Be *atomic*, capturing the state of all files at exactly the same time.

  *If you are not sure that the snapshot technology is atomic, do not use it. Use the **dsbackup** command instead.*

- Capture the state of all data (`db/`) and (`changelogDb/`) changelog files together.

  When using a file system-level snapshot feature, for example, keep at least all data and changelog files on the same file system. This is the case in a default server setup.

- Be paired with a specific server configuration.

  A snapshot of all files includes configuration files that may be specific to one DS server, and cannot be restored safely on another DS server with a different configuration. If you restore all system files, this principle applies to system configuration as well.

For details on making DS configuration files as generic as possible, see Property Value Substitution.

If snapshots in your deployment do not meet these criteria, *you must stop the DS server before taking the snapshot*. You must also take care not to restore incompatible configuration files.

Restore

IMPORTANT

After you restore a replicated backend, replication brings it up to date with changes newer than the backup. Replication uses internal change log records to determine which changes to apply. This process happens *even if you only have a single server* that you configured for replication at setup time (by setting the replication port with the `--replicationPort` *`port`* option). To prevent replication from replaying changes newer than the backup you restore, refer to Disaster recovery.

Replication purges internal change log records, however, to prevent the change log from growing indefinitely. Replication can only bring the backend up to date if the change log still includes the last change backed up.

For this reason, when you restore a replicated backend from backup, *the backup must be newer than the last purge of the replication change log (default: 3 days).*

If no backups are newer than the replication purge delay, do not restore from a backup. Initialize the replica instead, without using a backup. For details, see Manual Initialization.

Restore Data (Server Task)

1. Verify the backup you intend to restore.

   The following example verifies the most recent backup of the `dsEvaluation` backend:

   ```
   $ dsbackup \
    list \
    --backupLocation bak \
    --backendName dsEvaluation \
    --last \
    --verify
   ```

2. Schedule the restore operation as a task, binding as a user with the `backend-restore` administrative privilege.

The following example schedules an immediate restore task for the `dsEvaluation` backend:

```
$ dsbackup \
 restore \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --backupLocation bak \
 --backendName dsEvaluation
```

To restore the latest backups of more than one backend, specify the `--backendName` option multiple times.

To restore a specific backup, specify the `--backupId` option. To restore multiple specific backups of different backends, specify the `--backupId` option multiple times.

To list backup information without performing verification, use the **`dsbackup list`** command without the `--verify` option. The output includes backup IDs for use with the `--backupId` option.

For details, see dsbackup.

Restore Data (External Command)

1. Stop the server if it is running:

   ```
   $ stop-ds --quiet
   ```

2. Verify the backup you intend to restore.

   The following example verifies the most recent backup of the `dsEvaluation` backend:

   ```
   $ dsbackup \
    list \
    --backupLocation bak \
    --backendName dsEvaluation \
   ```

```
 --last \
 --verify
```

3. Restore using the `--offline` option.

   The following example restores the `dsEvaluation` backend:

   ```
   $ dsbackup \
    restore \
    --offline \
    --backupLocation bak \
    --backendName dsEvaluation
   ```

   To restore the latest backups of more than one backend, specify the `--backendName` option multiple times.

   To restore a specific backup, specify the `--backupId` option. To restore multiple specific backups of different backends, specify the `--backupId` option multiple times.

   To list backup information without performing verification, use the **`dsbackup list`** command without the `--verify` option. The output includes backup IDs for use with the `--backupId` option.

   For details, see dsbackup.

4. Start the server:

   ```
   $ start-ds --quiet
   ```

Restore Configuration Files

1. Stop the server:

   ```
   $ stop-ds --quiet
   ```

2. Restore the configuration files from the backup, overwriting existing files:

   ```
   $ tar -zxvf backup-config-<date>.tar.gz
   ```

3. Start the server:

```
$ start-ds --quiet
```

Restore From a Snapshot

ForgeRock recommends using the **dsbackup** command when possible for backup and restore operations.

You can use snapshot technology as an alternative to the **dsbackup** command, but you must be careful how you use it. For details, see Back Up Using Snapshots.

Take the following points into account before restoring a snapshot:

- When you restore files for a replicated backend, *the snapshot must be newer than the last purge of the replication change log (default: 3 days).*
- Stop the DS server before you restore the files.
- The DS configuration files in the snapshot must match the configuration where you restore the snapshot.

  If the configuration uses expressions, define their values for the current server before starting DS.
- When using snapshot files to initialize replication, only restore the data (`db/`) files for the target backend.

  Depending on the snapshot technology, you might need to restore the files separately, and then move only the target backend files from the restored snapshot.
- When using snapshot files to restore replicated data to a known state, stop all affected servers before you restore.

Purge Old Files

Periodically purge old backup files with the **dsbackup purge** command. The following example removes all backup files older than the default replication purge delay:

```
$ dsbackup \
 purge \
 --offline \
 --backupLocation bak \
 --olderThan 3d
```

This example runs the external command without contacting the server process. You can also purge backups by ID, or by backend name, and you can specify the number of backups to keep. For details, see dsbackup.

To purge files as a server task, use the task options, such as `--recurringTask` . The user must have the `backend-backup` administrative privilege to schedule a purge task.

Cloud Storage

You can stream backup files to cloud storage, and restore them directly from cloud storage.

The implementation supports these providers:

- Amazon AWS S3
- Azure Cloud Storage
- Google Cloud Storage

Follow these steps to store backup files in the cloud:

1. Get a storage account and space from the cloud provider where the server can store backup files.

   This storage space is referred to below as *cloud-bak*.
2. Get credentials from the cloud provider.

   The DS server backing up files must have read, write, and delete access. For information about granting access, see the access control documentation for your provider.

   If you are not yet familiar with cloud storage, see the documentation from your provider for help. The following table provides links to the documentation for supported providers:

Provider	Hints
Amazon AWS S3	For details on setting up S3 and working with S3 buckets, see the Amazon Web Services documentation on Getting started with Amazon Simple Storage Service.
Azure Cloud Storage	DS authenticates to Azure with an Azure storage account. For details, see the Microsoft documentation on how to Create an Azure Storage account, or to Create a BlockBlobStorage account.

Provider	Hints
Google Cloud Storage	DS authenticates to Google Cloud with a service account. For details, see the Google documentation on Getting Started with Authentication. For details about creating and managing storage buckets, see the Google How-To documentation on Creating buckets, and Working with buckets.

3. Set environment variables for the credentials:

Provider	Environment Variable(s)
Amazon AWS S3	`export AWS_ACCESS_KEY_ID=`*`aws-access-key`* `export AWS_SECRET_ACCESS_KEY=`*`aws-secret-key`*
Azure Cloud Storage	`export AZURE_ACCOUNT_NAME=`*`azure-account-name`* `export AZURE_ACCOUNT_KEY=`*`azure-account-key`*
Google Cloud Storage	`export GOOGLE_CREDENTIALS=`*`/path/to/gcp-credentials.json`* (optional)

4. Restart the DS server so that it reads the environment variables you set:

```
$ stop-ds --restart
```

5. Run **dsbackup** commands with all required provider-specific options.

   The options in the following table use the providers' default storage endpoints:

Provider	Required Options

Amazon AWS S3	`--storageProperty s3.keyId.env.var:AWS_ACCESS_KEY_ID \`
	`--storageProperty s3.secret.env.var:AWS_SECRET_ACCESS_KEY \`
	`--backupLocation s3://cloud-bak`

Provider	Required Options
Azure Cloud Storage	`--storageProperty az.accountName.env.var:AZURE_ACCOUNT_NAME \ --storageProperty az.accountKey.env.var:AZURE_ACCOUNT_KEY \ --backupLocation az://cloud-bak`

Google Cloud Storage	`--storageProperty gs.credentials.path:/path/to/gcp-credentials.json \ --backupLocation gs://cloud-bak`
	or
	`--storageProperty gs.credentials.env.var:GOOGLE_CREDENTIALS \ --backupLocation gs://cloud-bak`

If your cloud storage *does not use the default endpoint*, add one of the following options:

`--storage-property endpoint:`*`endpoint-url`*

`--storage-property endpoint.env.var:`*`environment-variable-for-endpoint-url`*

For Azure cloud storage, the *endpoint-url* starts with the account name. Examples include `https://`*`azure-account-name`*`.blob.core.windows.net`, `https://${AZURE_ACCOUNT_NAME}.blob.core.windows.net`, and `https://${AZURE_ACCOUNT_NAME}.some.private.azure.endpoint`.

Cloud storage requires working space in the local system temporary directory. Some cloud storage providers require sending the content length with each file.

To send the correct content length, the **`dsbackup`** command writes each prepared backup file to the system temporary directory before upload. It deletes each file after successful upload.

Cloud storage samples

Click the samples for your storage provider to expand the section and see the commands:

▼ AWS samples

```
#
# API keys created through the AWS API gateway console:
#
export AWS_ACCESS_KEY_ID=aws-access-key-id
export AWS_SECRET_ACCESS_KEY=aws-secret-key

# These samples use the following S3 bucket, and a non-default
endpoint:
# S3 bucket: s3://ds-test-backup
# S3 endpoint: https://s3.us-east-1.amazonaws.com

# Back up the dsEvaluation backend offline:
dsbackup create --backendName dsEvaluation --offline \
 --backupLocation s3://ds-test-backup \
 --storageProperty s3.keyId.env.var:AWS_ACCESS_KEY_ID \
 --storageProperty s3.secret.env.var:AWS_SECRET_ACCESS_KEY \
 --storageProperty endpoint:https://s3.us-east-1.amazonaws.com

# List and verify the latest backup files for each backend at
this location:
dsbackup list --verify --last \
 --backupLocation s3://ds-test-backup \
 --storageProperty s3.keyId.env.var:AWS_ACCESS_KEY_ID \
 --storageProperty s3.secret.env.var:AWS_SECRET_ACCESS_KEY \
 --storageProperty endpoint:https://s3.us-east-1.amazonaws.com

# Restore dsEvaluation from backup offline:
dsbackup restore --backendName dsEvaluation --offline \
 --backupLocation s3://ds-test-backup \
 --storageProperty s3.keyId.env.var:AWS_ACCESS_KEY_ID \
 --storageProperty s3.secret.env.var:AWS_SECRET_ACCESS_KEY \
 --storageProperty endpoint:https://s3.us-east-1.amazonaws.com

# Purge all dsEvaluation backup files:
dsbackup purge --backendName dsEvaluation --keepCount 0 --
offline \
 --backupLocation s3://ds-test-backup \
 --storageProperty s3.keyId.env.var:AWS_ACCESS_KEY_ID \
 --storageProperty s3.secret.env.var:AWS_SECRET_ACCESS_KEY \
 --storageProperty endpoint:https://s3.us-east-1.amazonaws.com
```

▼ Azure samples

```
#
# Credentials for Azure storage, where the Azure account is
found in key1 in the Azure console:
#
export AZURE_ACCOUNT_NAME=azure-account-name
export AZURE_ACCOUNT_KEY=azure-account-key

# These samples use the following Azure storage, and a non-
default endpoint:
# Azure storage: az://ds-test-backup/test1
# Azure endpoint:
https://${AZURE_ACCOUNT_NAME}.blob.core.windows.net

# Back up the dsEvaluation backend offline:
dsbackup create --backendName dsEvaluation --offline \
 --backupLocation az://ds-test-backup/test1 \
 --storageProperty az.accountName.env.var:AZURE_ACCOUNT_NAME \
 --storageProperty az.accountKey.env.var:AZURE_ACCOUNT_KEY \
 --storageProperty
"endpoint:https://${AZURE_ACCOUNT_NAME}.blob.core.windows.net"

# List and verify the latest backup files for each backend at
this location:
dsbackup list --verify --last \
 --backupLocation az://ds-test-backup/test1 \
 --storageProperty az.accountName.env.var:AZURE_ACCOUNT_NAME \
 --storageProperty az.accountKey.env.var:AZURE_ACCOUNT_KEY \
 --storageProperty
"endpoint:https://${AZURE_ACCOUNT_NAME}.blob.core.windows.net"

# Restore dsEvaluation from backup offline:
dsbackup restore --backendName dsEvaluation --offline \
 --backupLocation az://ds-test-backup/test1 \
 --storageProperty az.accountName.env.var:AZURE_ACCOUNT_NAME \
 --storageProperty az.accountKey.env.var:AZURE_ACCOUNT_KEY \
 --storageProperty
"endpoint:https://${AZURE_ACCOUNT_NAME}.blob.core.windows.net"

# Purge all dsEvaluation backup files:
dsbackup purge --backendName dsEvaluation --keepCount 0 --
offline \
 --backupLocation az://ds-test-backup/test1 \
 --storageProperty az.accountName.env.var:AZURE_ACCOUNT_NAME \
 --storageProperty az.accountKey.env.var:AZURE_ACCOUNT_KEY \
```

```
 --storageProperty
"endpoint:https://${AZURE_ACCOUNT_NAME}.blob.core.windows.net"
```

▼ Google cloud samples

```
#
# Credentials generated with and download from the Google cloud
console:
#
export GOOGLE_CREDENTIALS=/path/to/gcp-credentials.json

# These samples use the following cloud storage, and endpoint:
# Google storage: gs://ds-test-backup/test1
# Google endpoint: https://www.googleapis.com

# Back up the dsEvaluation backend offline:
dsbackup create --backendName dsEvaluation --offline \
 --backupLocation gs://ds-test-backup/test1 \
 --storageProperty gs.credentials.env.var:GOOGLE_CREDENTIALS \
 --storageProperty endpoint:https://www.googleapis.com

# List and verify the latest backup files for each backend at
this location:
dsbackup list --verify --last \
 --backupLocation gs://ds-test-backup/test1 \
 --storageProperty gs.credentials.env.var:GOOGLE_CREDENTIALS \
 --storageProperty endpoint:https://www.googleapis.com

# Restore dsEvaluation from backup offline:
dsbackup restore --backendName dsEvaluation --offline \
 --backupLocation gs://ds-test-backup/test1 \
 --storageProperty gs.credentials.env.var:GOOGLE_CREDENTIALS \
 --storageProperty endpoint:https://www.googleapis.com

# Purge all dsEvaluation backup files:
dsbackup purge --backendName dsEvaluation --keepCount 0 --
offline \
 --backupLocation gs://ds-test-backup/test1 \
 --storageProperty gs.credentials.env.var:GOOGLE_CREDENTIALS \
 --storageProperty endpoint:https://www.googleapis.com
```

Disaster recovery

Directory services are critical to authentication, session management, authorization, and more. When directory services are broken, quick recovery is a must.

In DS directory services, a *disaster* is a serious data problem affecting the entire replication topology. Replication can't help you recover from a disaster because it replays data changes everywhere.

Disaster recovery comes with a service interruption, the loss of recent changes, and a reset for replication. It is rational in the event of a real disaster. It's unnecessary to follow the disaster recovery procedure for a hardware failure or a server that's been offline too long and needs reinitialization. Even if you lose most of your DS servers, you can still rebuild the service without interruption or data loss.

IMPORTANT

For disaster recovery to be quick, you must prepare in advance.

Don't go to production until you have successfully tested your disaster recovery procedures.

The following example helps prepare to recover from a disaster. It shows the following tasks:

- Back up a DS directory service.
- Restore the service to a known state.
- Validate the procedure.

Tasks

The following tasks demonstrate a disaster recovery procedure on a single computer two replicated DS servers set up for evaluation.

In deployment, the procedure involves multiple computers, but the order and content of the tasks remain the same. Before you perform the procedure in production, make sure you have copies of the following:

- The deployment description, documentation, plans, runbooks, and scripts.
- The system configuration and software, including the Java installation.
- The DS software and any customizations, plugins, or extensions.
- A recent backup of any external secrets required, such as an HSM or a CA key.
- A recent backup of each server's configuration files, matching the production configuration.
- The deployment ID and password.

IMPORTANT

IMPORTANT

This procedure applies to DS versions providing the **`dsrepl disaster-recovery`** command.

For deployments with any earlier DS servers that don't provide the command, you can't use this procedure. Instead, refer to How do I perform disaster recovery steps in DS (All versions)?

Disaster recovery has these characteristics:

- You perform disaster recovery on a stopped server, one server at a time.
- Disaster recovery is per base DN, like replication.
- On each server you recover, you use the same *disaster recovery ID*, a unique identifier for this recovery.

To minimize the service interruption, this example recovers the servers one by one. It is also possible to perform disaster recovery in parallel by stopping and starting all servers together.

Task 1: Back up directory data

Back up data while the directory service is running smoothly. For additional details, refer to Backup and Restore.

1. Back up the directory data.

   The following command backs up directory data created for evaluation:

   ```
   $ /path/to/opendj/bin/dsbackup \
    create \
    --start 0 \
    --backupLocation /path/to/opendj/bak \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePassword:file /path/to/opendj/config/keystore.pin
   ```

   The command returns, and the DS server runs the backup task in the background.

   When adapting the recovery process for deployment, schedule a backup task to run regularly for each database backend.

2. Check the backup task finishes successfully:

```
$ /path/to/opendj/bin/manage-tasks \
 --summary \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --no-prompt
```

The status of the backup task is "Completed successfully" when it is done.

Recovery from disaster means stopping the directory service and losing the latest changes. The more recent the backup, the fewer changes you lose during recovery. Backup operations are cumulative, so you can schedule them regularly without using too much disk space as long as you purge outdated backup files. As you script your disaster recovery procedures for deployment, schedule a recurring backup task to have safe, current, and complete backup files for each backend.

Task 2: Recover from a disaster

This task restores the directory data from backup files created before the disaster. Adapt this procedure as necessary if you have multiple directory backends to recover.

IMPORTANT

All changes since the last backup operation are lost.

Subtasks:

- Prepare for recovery
- Recover the first directory server
- Recover remaining servers

Prepare for recovery

1. If you have lost DS servers, replace them with servers configured as before the disaster.

   In this example, no servers were lost. Reuse the existing servers.

2. On each replica, prevent applications from making changes to the backend for the affected base DN. Changes made during recovery would be lost or could not be replicated:

```
$ /path/to/opendj/bin/dsconfig \
 set-backend-prop \
 --backend-name dsEvaluation \
 --set writability-mode:internal-only \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --no-prompt

$ /path/to/replica/bin/dsconfig \
 set-backend-prop \
 --backend-name dsEvaluation \
 --set writability-mode:internal-only \
 --hostname localhost \
 --port 14444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --no-prompt
```

In this example, the first server's administrative port is `4444` . The second server's administrative port is `14444` .

Recover the first directory server

IMPORTANT

IMPORTANT

DS uses the *disaster recovery ID* to set the *generation ID*, an internal, shorthand form of the initial replication state. Replication only works when the data for the base DN share the same generation ID on each server.

There are two approaches to using the **`dsrepl disaster-recovery`** command. Use one or the other:

- (*Recommended*) Let DS generate the disaster recovery ID on a first replica. Use the generated ID on all other servers you recover.

  When you use the generated ID, the **`dsrepl disaster-recovery`** command verifies each server you recover has the same initial replication state as the first server.

- Use the recovery ID of your choice on all servers.

  Don't use this approach if the replication topology includes one or more standalone replication servers. It won't work.

  This approach works when you can't define a "first" replica, for example, because you've automated the recovery process in an environment where the order of recovery is not deterministic.

  When you choose the recovery ID, the **`dsrepl disaster-recovery`** command *doesn't* verify the data match. The command uses your ID as the random seed when calculating the new generation ID. For the new generation IDs to match, your process must have restored the same data on each server. Otherwise, replication won't work between servers whose data does not match.

  If you opt for this approach, skip these steps. Instead, proceed to Recover remaining servers.

Don't mix the two approaches in the same disaster recovery procedure. Use the generated recovery ID or the recovery ID of your choice, but do not use both.

This process generates the disaster recovery ID to use when recovering the other servers.

1. Stop the directory server you use to start the recovery process:

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

2. Restore the affected data on this directory server:

```
$ /path/to/opendj/bin/dsbackup \
 restore \
```

```
 --offline \
 --backendName dsEvaluation \
 --backupLocation /path/to/opendj/bak
```

Changes to the affected data that happened after the backup are lost. Use the most recent backup files prior to the disaster.

TIP

This approach to restoring data works in deployments with the same DS server version. When all DS servers share the same DS version, you can restore all the DS directory servers from the same backup data.

Backup archives are *not guaranteed to be compatible* across major and minor server releases. *Restore backups only on directory servers of the same major or minor version.*

3. Run the command to begin the disaster recovery process.

   When this command completes successfully, it displays the disaster recovery ID:

```
$ /path/to/opendj/bin/dsrepl \
 disaster-recovery \
 --baseDn dc=example,dc=com \
 --generate-recovery-id \
 --no-prompt
Disaster recovery id: <generatedId>
```

   Record the <generatedId>. You will use it to recover all other servers.

4. Start the recovered server:

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

5. Test the data you restored is what you expect.

6. Start backing up the recovered directory data.

   As explained in New backup after recovery, you can no longer rely on pre-recovery backup data after disaster recovery.

7. Allow external applications to make changes to directory data again:

```
$ /path/to/opendj/bin/dsconfig \
 set-backend-prop \
 --backend-name dsEvaluation \
 --set writability-mode:enabled \
 --hostname localhost \
```

```
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --no-prompt
```

You have recovered this replica and begun to bring the service back online. To enable replication with other servers to resume, recover the remaining servers.

Recover remaining servers

IMPORTANT

Make sure you have a disaster recovery ID. Use the same ID for all DS servers in this recovery procedure:

- (*Recommended*) If you generated the ID as described in Recover the first directory server, use it.
- If not, use a unique ID of your choosing for this recovery procedure.

  For example, you could use the date at the time you begin the procedure.

You can perform this procedure in parallel on all remaining servers or on one server at a time. For each server:

1. Stop the server:

   ```
   $ /path/to/replica/bin/stop-ds
   ```

2. Unless the server is a standalone replication server, restore the affected data:

   ```
   $ /path/to/replica/bin/dsbackup \
    restore \
    --offline \
    --backendName dsEvaluation \
    --backupLocation /path/to/opendj/bak
   ```

3. Run the recovery command.

   The following command uses a generated ID. It verifies this server's data matches the first server you recovered:

   ```
   $ export DR_ID=<generatedId>
   $ /path/to/replica/bin/dsrepl \
   ```

```
 disaster-recovery \
 --baseDn dc=example,dc=com \
 --generated-id ${DR_ID} \
 --no-prompt
```

If the recovery ID is a unique ID of your choosing, use **`dsrepl disaster-recovery --baseDn <base-dn> --user-generated-id <recoveryId>`** instead. This alternative doesn't verify the data on each replica match and won't work if the replication topology includes one or more standalone replication servers.

4. Start the recovered server:

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

5. If this is a directory server, test the data you restored is what you expect.
6. If this is a directory server, allow external applications to make changes to directory data again:

```
$ /path/to/replica/bin/dsconfig \
 set-backend-prop \
 --backend-name dsEvaluation \
 --set writability-mode:enabled \
 --hostname localhost \
 --port 14444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --no-prompt
```

After completing these steps for all servers, you have restored the directory service and recovered from the disaster.

Validation

After recovering from the disaster, validate replication works as expected. Use the following steps as a simple guide.

1. Modify an entry on one replica.

   The following command updates Babs Jensen's description to `Post recovery`:

```
$ /path/to/opendj/bin/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: Post recovery
EOF
# MODIFY operation successful for DN
uid=bjensen,ou=People,dc=example,dc=com
```

2. Read the modified entry on another replica:

```
$ /path/to/replica/bin/ldapsearch \
 --hostname localhost \
 --port 11636 \
 --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 \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 description
dn: uid=bjensen,ou=People,dc=example,dc=com
description: Post recovery
```

You have shown the recovery procedure succeeded.

Before deployment

When planning to deploy disaster recovery procedures, take these topics into account.

Recover before the purge delay

When recovering from backup, you must complete the recovery procedure while the backup is newer than the replication delay.

If this is not possible for all servers, recreate the remaining servers from scratch after recovering as many servers as possible and taking a new backup.

New backup after recovery

Disaster recovery resets the replication generation ID to a different format than you get when importing new directory data.

After disaster recovery, you can no longer use existing backup files for the recovered base DN. Directory servers can only replicate data under a base DN with directory servers having the same generation ID. The old backups no longer have the right generation IDs.

Instead, immediately after recovery, back up data from the recovered base DN and use the new backups going forward.

You can purge older backup files to prevent someone accidentally restoring from a backup with an outdated generation ID.

Change notifications reset

Disaster recovery clears the changelog for the recovered base DN.

If you use change number indexing for the recovered base DN, disaster recovery resets the change number.

Standalone servers

If you have standalone replication servers and directory servers, you might not want to recover them all at once.

Instead, in each region, alternate between recovering a standalone directory server then a standalone replication server to reduce the time to recovery.

Reference material

Reference	Description
About replication	In-depth introduction to replication concepts
Backup and Restore	The basics, plus backing up to the cloud and using filesystem snapshots
Cryptographic Keys	About keys, including those for encrypting and decrypting backup files
Data Storage	Details about exporting and importing LDIF, common data stores

Accounts

Account Lockout

Account lockout settings are part of password policy. The server locks an account after the specified number of consecutive authentication failures. For example, users are allowed three consecutive failures before being locked out for five minutes. Failures themselves expire after five minutes.

The aim of account lockout is not to punish users who mistype their passwords. It protects the directory when an attacker attempts to guess a user password with repeated attempts to bind.

NOTE

Account lockout is not transactional across a replication topology. Under normal circumstances, replication propagates lockout quickly. If replication is ever delayed, an attacker with direct access to multiple replicas could try to authenticate up to the specified number of times on each replica before being locked out on all replicas.

The following command adds a replicated password policy to activate lockout:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=Lock after three failures,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: ds-pwp-password-policy
cn: Lock after three failures
ds-pwp-password-attribute: userPassword
ds-pwp-default-password-storage-scheme: PBKDF2-HMAC-SHA256
ds-pwp-lockout-failure-expiration-interval: 5 m
ds-pwp-lockout-duration: 5 m
ds-pwp-lockout-failure-count: 3
subtreeSpecification: { base "ou=people" }
EOF
```

Users with this policy are locked out after three failed attempts in succession:

```
$ ldapsearch \
 --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 \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail

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

$ ldapsearch \
 --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 fatfngrs \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail

The LDAP bind request failed: 49 (Invalid Credentials)

$ ldapsearch \
 --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 fatfngrs \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail

The LDAP bind request failed: 49 (Invalid Credentials)
```

```
$ ldapsearch \
 --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 fatfngrs \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail

The LDAP bind request failed: 49 (Invalid Credentials)

$ ldapsearch \
 --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 \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail

The LDAP bind request failed: 49 (Invalid Credentials)
```

Account Management

Disable an Account

1. Make sure the user running the **`manage-account`** command has access to perform the appropriate operations.

   Kirsten Vaughan is a member of the Directory Administrators group. For this example, she must have the `password-reset` privilege, and access to edit user attributes and operational attributes:

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

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

dn: ou=People,dc=example,dc=com
changetype: modify
add: aci
aci: (target="ldap:///ou=People,dc=example,dc=com")
(targetattr ="*||+")
 (version 3.0;acl "Admins can run amok"; allow(all)
  groupdn = "ldap:///cn=Directory
Administrators,ou=Groups,dc=example,dc=com";)
EOF
```

Notice here that the directory superuser, `uid=admin`, assigns privileges. Any administrator with the `privilege-change` privilege can assign privileges. However, if the administrator can update administrator privileges, they can assign themselves the `bypass-acl` privilege. Then they are no longer bound by access control instructions, including both user data ACIs and global ACIs. For this reason, do not assign the `privilege-change` privilege to normal administrator users.

2. Set the account status to disabled:

```
$ manage-account \
 set-account-is-disabled \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --operationValue true \
 --targetDN uid=bjensen,ou=people,dc=example,dc=com \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin

Account Is Disabled:  true
```

Activate a Disabled Account

1. Clear the disabled status:

```
$ manage-account \
 set-account-is-disabled \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --operationValue false \
 --targetDN uid=bjensen,ou=people,dc=example,dc=com \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin

Account Is Disabled:  false
```

Account Status Notifications

DS servers can send mail about account status changes. The DS server needs an SMTP server to send messages, and needs templates for the mail it sends. By default, message templates are in English, and found in the `/path/to/opendj/config/messages/` directory.

DS servers generate notifications only when the server writes to an entry or evaluates a user entry for authentication. A server generates account enabled and account disabled notifications when the user account is enabled or disabled with the **`manage-account`** command. A server generates password expiration notifications when a user tries to bind.

For example, if you configure a notification for password expiration, that notification gets triggered when the user authenticates during the password expiration warning interval. The server does not automatically scan entries to send password expiry notifications.

DS servers implement controls that you can pass in an LDAP search to determine whether a user's password is about to expire. See Supported LDAP Controls for a list. Your script or client application can send notifications based on the results of the search.

Send Account Status Mail

1. Configure an SMTP server to use when sending messages:

```
$ dsconfig \
 create-mail-server \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --server-name "SMTP server" \
 --set enabled:true \
 --set auth-username:mail.user \
 --set auth-password:password \
 --set smtp-server:smtp.example.com:587 \
 --set trust-manager-provider:"JVM Trust Manager" \
 --set use-start-tls:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

2. Prepare the DS server to mail users about account status.

   The following example configures the server to send text-format mail messages:

```
$ dsconfig \
 set-account-status-notification-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name "SMTP Handler" \
 --set enabled:true \
 --set email-address-attribute-type:mail \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

   Notice that the server finds the user's mail address on the attribute on the user's entry, specified by `email-address-attribute-type` . You can also configure the `message-subject` and `message-template-file` properties. Use interactive mode to make the changes.

You find templates for messages by default under the `config/messages` directory. Edit the templates as necessary.

If you edit the templates to send HTML rather than text messages, then set the advanced property, `send-email-as-html`:

```
$ dsconfig \
 set-account-status-notification-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name "SMTP Handler" \
 --set enabled:true \
 --set send-email-as-html:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. Adjust applicable password policies to use the account status notification handler you configured:

```
$ dsconfig \
 set-password-policy-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set account-status-notification-handler:"SMTP Handler"
\
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

When configuring a subentry password policy, set the `ds-pwp-account-status-notification-handler` attribute, an attribute of the `ds-pwp-password-policy` object class.

Message Templates

When editing the `config/messages` templates, use the following tokens, which the server replaces with text:

%%notification-type%%
The name of the notification type.

%%notification-message%%
The message for the notification.

%%notification-user-dn%%
The string representation of the user DN that is the target of the notification.

%%notification-user-attr:attrname%%
The value of the attribute specified by attrname from the user's entry.

If the specified attribute has multiple values, then this is the first value encountered.
If the specified attribute does not have any values, then this is an empty string.

%%notification-property:propname%%
The value of the specified property.

If the specified property has multiple values, then this is the first value encountered.
If the specified property does not have any values, then this is an empty string.

Valid propname values include the following:

- `account-unlock-time`
- `new-password`
- `old-password`
- `password-expiration-time`
- `password-policy-dn`
- `seconds-until-expiration`
- `seconds-until-unlock`
- `time-until-expiration`
- `time-until-unlock`

Resource Limits

Search Limits

You can set limits on search operations:

- The *lookthrough limit* defines the maximum number of candidate entries that the DS server considers when processing a search.

  The default lookthrough limit of 5000 is set by the global server property `lookthrough-limit` .

You can override the limit per user with the operational attribute, `ds-rlim-lookthrough-limit`.

- The *size limit* sets the maximum number of entries returned for a search.

  The default size limit of 1000 is set by the global server property `size-limit`.

  You can override the limit per user with the operational attribute, `ds-rlim-size-limit`.

  Search requests can include a size limit setting. The **`ldapsearch`** command has a `--sizeLimit` option.

- The *time limit* defines the maximum processing time for a search operation.

  The default time limit of 1 minute is set by the global server property `time-limit`.

  You can override the limit on a per user basis with the operational attribute, `ds-rlim-time-limit`. Times for `ds-rlim-time-limit` are expressed in seconds.

  In addition, search requests themselves can include a time limit setting. The **`ldapsearch`** command has an `--timeLimit` option.

- The *idle time limit* defines how long an idle connection remains open.

  No default idle time limit is set. You can set an idle time limit by using the global server property `idle-time-limit`.

  You can override the limit on a per user basis with the operational attribute, `ds-rlim-idle-time-limit`. Times for `ds-rlim-idle-time-limit` are expressed in seconds.

- The maximum number of persistent searches is set by the global server property `max-psearches`.

Set Limits For a User

1. Give an administrator access to update the operational attributes related to search limits:

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

```
dn: ou=People,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "ds-rlim-lookthrough-limit||ds-rlim-
time-limit||ds-rlim-size-limit")
 (version 3.0;acl "Allow Kirsten Vaughan to manage search
limits";
 allow (all) (userdn =
"ldap:///uid=kvaughan,ou=People,dc=example,dc=com");)
EOF
```

2. Change the user entry to set the limits to override:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: ds-rlim-size-limit
ds-rlim-size-limit: 10
EOF
```

When Babs Jensen performs an indexed search returning more than 10 entries, she sees the following message:

```
$ ldapsearch \
 --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 \
 --baseDN dc=example,dc=com \
 "(sn=jensen)"

# The LDAP search request failed: 4 (Size Limit Exceeded)
```

```
# Additional Information:  This search operation has sent
the maximum of 10 entries to the client
```

Set Limits For Users in a Group

1. Give an administrator the privilege to write subentries:

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

   Notice here that the directory superuser, `uid=admin` , assigns privileges. Any administrator with the `privilege-change` privilege can assign privileges. However, if the administrator can update administrator privileges, they can assign themselves the `bypass-acl` privilege. Then they are no longer bound by access control instructions, including both user data ACIs and global ACIs. For this reason, do not assign the `privilege-change` privilege to normal administrator users.

2. Create an LDAP subentry to specify the limits using collective attributes:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: cn=Remove Administrator Search
Limits,dc=example,dc=com
```

```
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Remove Administrator Search Limits
ds-rlim-lookthrough-limit;collective: 0
ds-rlim-size-limit;collective: 0
ds-rlim-time-limit;collective: 0
subtreeSpecification: {base "ou=people",
specificationFilter
  "(isMemberOf=cn=Directory
Administrators,ou=Groups,dc=example,dc=com)" }
EOF
```

The `base` entry identifies the branch that holds administrator entries. For details on how subentries apply, see About Subentry Scope.

3. Check the results:

```
$ 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 uid=kvaughan,ou=people,dc=example,dc=com \
 --searchScope base \
 "(&)" \
 ds-rlim-lookthrough-limit ds-rlim-time-limit ds-rlim-
size-limit

dn: uid=kvaughan,ou=People,dc=example,dc=com
ds-rlim-lookthrough-limit: 0
ds-rlim-size-limit: 0
ds-rlim-time-limit: 0
```

Limit Persistent Searches

An LDAP persistent search maintains an open a connection that may be be idle for long periods of time. Whenever a modification changes data in the search scope, the server

returns a search result. The more concurrent persistent searches, the more work the server has to do for each modification:

1. Set the global property `max-psearches` to limit total concurrent persistent searches.

   The following example limits the maximum number of persistent searchees to 30:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --set max-psearches:30 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

Connection Limits

Limit Total Connections

Each connection uses memory. On UNIX and Linux systems, each connection uses an available file descriptor.

To limit the total number of concurrent client connections that the server accepts, use the global setting `max-allowed-client-connections` . The following example sets the limit to 64K. 64K is the minimum number of file descriptors that should be available to the DS server:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --set max-allowed-client-connections:65536 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
```

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

Restrict Who Can Connect

To restrict which clients can connect to the server, use the global setting `allowed-client`, or `denied-client`. The following example restricts access to clients from the `example.com` domain:

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

Set these properties per `Connection Handler`. The settings on a connection handler override the global settings.

Limit Connections Per Client

To limit the number of concurrent connections from a client, use the global settings `restricted-client`, and `restricted-client-connection-limit`. The following example sets the limit for all clients on the `10.0.0.*` network to 1000 concurrent connections:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --set restricted-client:"10.0.0.*" \
 --set restricted-client-connection-limit:1000 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Set these properties per `Connection Handler`. The settings on a connection handler override the global settings.

The server applies the properties in this order:

1. If the `denied-client` property is set, the server denies connections from any client matching the settings.
2. If the `restricted-client` property is set, the server checks the number of connections from any client matching the settings.

   If a matching client exceeds `restricted-client-connection-limit` connections, the server refuses additional connections.
3. If the `allowed-client` property is set, the server allows connections from any client matching the settings.
4. If none of the properties are set, the server allows connections from any client.

Idle Time Limits

If client applications leave connections idle for long periods, you can drop their connections by setting the global configuration property idle-time-limit. By default, no idle time limit is set.

If your network is configured to drop connections that have been idle for some time, set the DS idle time limit to a lower value than the idle time limit for the network. This helps to ensure that idle connections are shut down in orderly fashion. Setting the DS limit lower than the network limit is particularly useful with networks that drop idle connections without cleanly closing the connection and notifying the client and server.

NOTE

DS servers do not enforce idle timeout for persistent searches.

The following example sets the idle-time-limit to 24 hours:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --set idle-time-limit:24h \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Request Size Limits

The default maximum request size is 5 MB. This is sufficient for most deployments. In cases where clients add groups with large numbers of members, requests can exceed the 5 MB limit.

The following example increases the limit to 20 MB for the LDAP connection handler:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name LDAP \
 --set max-request-size:20mb \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

This setting affects only the size of requests, not responses.

Limits and Proxied Authorization

Proxied authorization lets an application bind as one user and carry out LDAP operations on behalf of other users.

Resource limits do not change when the user proxies as another user. In other words, resource limits depend on the bind DN, not the proxy authorization identity.

Move a Server

The following procedure moves a server to the new host `new-server.example.com`. The steps skip creation of system accounts, startup scripts, and registration as a Windows service:

1. Stop the server:

   ```
   $ stop-ds
   ```

2. Renew the server certificate to account for the new hostname.

Skip this step if the server certificate is a wildcard certificate that is already valid for the new hostname.

The following command renews the server certificate generated with a deployment key:

```
$ dskeymgr \
 create-tls-key-pair \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --keyStoreFile /path/to/opendj/config/keystore \
 --keyStorePassword:file
/path/to/opendj/config/keystore.pin \
 --hostname localhost \
 --hostname new-server.example.com \
 --subjectDn CN=DS,O=ForgeRock
```

For more command options, refer to dskeymgr. The default validity for the certificate is one year.

3. Find and replace the old hostname with the new hostname in the server's configuration file, `config/config.ldif`.

   The following list includes configuration settings that may specify the server hostname:

   - `ds-cfg-advertised-listen-address`
   - `ds-cfg-bootstrap-replication-server`
   - `ds-cfg-listen-address`
   - `ds-cfg-server-fqdn`
   - `ds-cfg-source-address`

4. Move all files in the `/path/to/opendj` directory to the new server.
5. Start the server:

   ```
   $ start-ds
   ```

6. If the server you moved is referenced by others as a replication bootstrap server, update the replication bootstrap server configuration on those servers.

Performance Tuning

Performance Requirements

Your key performance requirement is to satisfy your users or customers with the resources available to you. Before you can solve potential performance problems, define what those users or customers expect. Determine which resources you will have to satisfy their expectations.

Service Level Objectives

A service level objective (SLO) is a target for a directory service level that you can measure quantitatively. If possible, base SLOs on what your key users expect from the service in terms of performance.

Define SLOs for at least the following areas:

- Directory service *response times*

  Directory service response times range from less than a millisecond on average, across a low latency connection on the same network, to however long it takes your network to deliver the response.

  More important than average or best response times is the response time distribution, because applications set timeouts based on worst case scenarios.

  An example response time performance requirement is, *Directory response times must average less than 10 milliseconds for all operations except searches returning more than 10 entries, with 99.9% of response times under 40 milliseconds.*

- Directory service *throughput*

  Directories can serve many thousands of operations per second. In fact there is no upper limit for read operations such as searches, because only write operations must be replicated. To increase read throughput, simply add additional replicas.

  More important than average throughput is peak throughput. You might have peak write throughput in the middle of the night when batch jobs update entries in bulk, and peak binds for a special event or first thing Monday morning.

  An example throughput performance requirement is, *The directory service must sustain a mix of 5,000 operations per second made up of 70% reads, 25% modifies, 3% adds, and 2% deletes.*

  Ideally, you mimic the behavior of key operations during performance testing, so that you understand the patterns of operations in the throughput you need to provide.

- Directory service *availability*

  DS software is designed to let you build directory services that are basically available, including during maintenance and even upgrade of individual servers.

To reach very high levels of availability, you must also ensure that your operations execute in a way that preserves availability.

Availability requirements can be as lax as a best effort, or as stringent as 99.999% or more uptime.

Replication is the DS feature that allows you to build a highly available directory service.

- Directory service administrative support

  Be sure to understand how you support your users when they run into trouble.

  While directory services can help you turn password management into a self-service visit to a web site, some users still need to know what they can expect if they need your help.

Creating an SLO, even if your first version consists of guesses, helps you reduce performance tuning from an open-ended project to a clear set of measurable goals for a manageable project with a definite outcome.

Resource Constraints

With your SLOs in hand, inventory the server, networks, storage, people, and other resources at your disposal. Now is the time to estimate whether it is possible to meet the requirements at all.

If, for example, you are expected to serve more throughput than the network can transfer, maintain high-availability with only one physical machine, store 100 GB of backups on a 50 GB partition, or provide 24/7 support all alone, no amount of tuning will fix the problem.

When checking that the resources you have at least theoretically suffice to meet your requirements, do not forget that high availability in particular requires at least two of everything to avoid single points of failure. Be sure to list the resources you expect to have, when and how long you expect to have them, and why you need them. Make note of what is missing and why.

Server Hardware

DS servers are pure Java applications, making them very portable. DS servers tend to perform best on single-board, x86 systems due to low memory latency.

Storage

High-performance storage is essential for handling high-write throughput. When the database stays fully cached in memory, directory read operations do not result in disk

I/O. Only writes result in disk I/O. You can further improve write performance by using solid-state disks for storage or file system cache.

WARNING

DS directory servers are designed to work with *local storage* for database backends. *Do not use network file systems, such as NFS, where there is no guarantee that a single process has access to files.*

Storage area networks (SANs) and attached storage are fine for use with DS directory servers.

Regarding database size on disk, sustained write traffic can cause the database to grow to more than twice its initial size on disk. This is normal behavior. The size on disk does not impact the DB cache size requirements.

To avoid directory database file corruption after crashes or power failures on Linux systems, enable file system write barriers, and make sure that the file system journaling mode is ordered. For details on how to enable write barriers and set the journaling mode for data, see the options for your file system in the **`mount`** command manual page.

Performance Tests

Even if you do not need high availability, you still need two of everything, because your test environment needs to mimic your production environment as closely as possible.

In your test environment, set up DS servers just as you do in production. Conduct experiments to determine how to best meet your SLOs.

The following command-line tools help with basic performance testing:

- The makeldif command generates sample data with great flexibility.
- The **`addrate`** command measures add and delete throughput and response time.
- The **`authrate`** command measures bind throughput and response time.
- The **`modrate`** command measures modification throughput and response time.
- The **`searchrate`** command measures search throughput and response time.

All **`*rate`** commands display response time distributions measurements, and support testing at specified levels of throughput.

For additional precision when evaluating response times, use the global configuration setting etime-resolution. To change elapsed processing time resolution from milliseconds (default) to nanoseconds:

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

The `etime` , recorded in the server access log, indicates the elapsed time to process the request. The `etime` starts when the decoded operation is available to be processed by a worker thread.

Test performance with your production-ready configuration. If, however, you simply want to demonstrate top performance, take the following points into account:

- Incorrect JVM tuning slows down server and tool performance. Make sure the JVM is tuned for best performance.

  For example, set the following environment variable, then restart the server and run the performance tools again to take the change into account:

  ```
  export OPENDJ_JAVA_ARGS="-XX:+UseParallelGC -
  XX:MaxTenuringThreshold=1"
  ```

  If the server heap is very large, see the details in Java Settings.
- Unfiltered access logs record messages for each client request. Turn off full access logging.

  For example, set `enabled:false` for the `Json File-Based Access Logger` log publisher, and any other unfiltered log publishers that are enabled.
- Secure connections are recommended, and they can be costly.

  Set `require-secure-authentication:false` in the password policies governing the bind entries, and bind using insecure connections.

Performance Settings

Use the following suggestions when your tests show that DS performance is lacking, even though you have the right underlying network, hardware, storage, and system resources in place.

Maximum Open Files

DS servers must open many file descriptors when handling thousands of client connections.

Linux systems often set a limit of 1024 per user. That setting is too low to accept thousands of client connections.

Make sure the server can use at least 64K (65536) file descriptors. For example, when running the server as user `opendj` on a Linux system that uses `/etc/security/limits.conf` to set user level limits, set soft and hard limits by adding these lines to the file:

```
opendj soft nofile 65536
opendj hard nofile 131072
```

The example above assumes the system has enough file descriptors available overall. Check the Linux system overall maximum as follows:

```
$ cat /proc/sys/fs/file-max
204252
```

Linux Page Caching

Default Linux virtual memory settings cause significant buildup of dirty data pages before flushing them. When the kernel finally flushes the pages to disk, the operation can exhaust the disk I/O for up to several seconds. Application operations waiting on the file system to synchronize to disk are blocked.

The default virtual memory settings can therefore cause DS server operations to block for seconds at a time. Symptoms included high outlier etimes, even for very low average etimes. For sustained high loads, such as import operations, the server has to maintain thousands of open file descriptors.

To avoid these problems, tune Linux page caching. As a starting point for testing and tuning, set `vm.dirty_background_bytes` to one quarter of the disk I/O per second, and `vm.dirty_expire_centisecs` to 1000 (10 seconds) using the **`sysctl`** command. This causes the kernel to flush more often, and limits the pauses to a maximum of 250 milliseconds.

For example, if the disk I/O is 80 MB/second for writes, the following example shows an appropriate starting point. It updates the `/etc/sysctl.conf` file to change the setting permanently, and uses the **`sysctl -p`** command to reload the settings:

```
$ echo vm.dirty_background_bytes=20971520 | sudo tee -a
/etc/sysctl.conf
[sudo] password for admin:

$ echo vm.dirty_expire_centisecs=1000 | sudo tee -a
/etc/sysctl.conf

$ sudo sysctl -p
vm.dirty_background_bytes = 20971520
vm.dirty_expire_centisecs = 1000
```

Be sure to test and adjust the settings for your deployment.

For additional details, see the Oracle documentation on Linux Page Cache Tuning, and the Linux **sysctl** command virtual memory kernel reference.

Java Settings

Default Java settings let you evaluate DS servers using limited system resources. For high performance production systems, test and run with a tuned JVM.

TIP

To apply JVM settings for a server, edit `config/java.properties`, and restart the server.

Availability of the following **java** options depends on the JVM:

-Xmx
: If you observe any internal node evictions, add more RAM to the system. If adding RAM is not an option, increase the maximum heap size to optimize RAM allocation. For details, see Cache Internal Nodes.

  Use at least a 2 GB heap unless your data set is small.

-XX:+DisableExplicitGC
: When using JMX, add this option to the list of `start-ds.java-args` arguments to avoid periodic full GC events.

  JMX is based on RMI, which uses references to objects. By default, the JMX client and server perform a full GC periodically to clean up stale references. As a result, the default settings cause JMX to cause a full GC every hour.

  Avoid using this argument with `import-ldif.offline.java-args` or when using the **import-ldif** command. The import process uses garbage collection to manage memory and references to memory-mapped files.

`-XX:MaxTenuringThreshold=1`
: This sets the maximum number of GC cycles an object stays in survivor spaces before it is promoted into the old generation space.

  Setting this option as suggested reduces the new generation GC frequency and duration. The JVM quickly promotes long-lived objects to the old generation space, rather than letting them accumulate in new generation survivor spaces, copying them for each GC cycle.

`-Xlog:gc=level:file`
: Log garbage collection messages when diagnosing JVM tuning problems. You can turn the option off when everything is running smoothly.

  Always specify the output file for the garbage collection log. Otherwise, the JVM logs the messages to the `opendj/logs/server.out` file, mixing them with other messages, such as stack traces from the **`supportextract`** command.

  For example, **`-Xlog:gc=info:file=/path/to/gc.log`** logs informational messages about garbage collection to the file, `/path/to/gc.log`.

  For details, use the **`java -Xlog:help`** command.

`-XX:TieredStopAtLevel=1`
: Short-lived client tools, such as the **`ldapsearch`** command, start up faster when this option is set to `1` as shown.

`-XX:+UseG1GC -XX:MaxGCPauseMillis=100`

Java 11
: Use G1 GC (the default) when the heap size is 8 GB or more.

Java 17
: Use G1 GC.

`-XX:+UseParallelGC`

Java 11
: Use parallel GC when the heap size is less than 8 GB.

Java 17
: Use G1 GC instead.

Data Storage Settings

By default, DS servers compress attribute descriptions and object class sets to reduce data size. This is called compact encoding.

By default, DS servers do not compress entries stored in its backend database. If your entries hold values that compress well, such as text, you can gain space. Set the backend

property `entries-compressed:true` , and reimport the data from LDIF. The DS server compresses entries before writing them to the database:

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

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

DS directory servers do not proactively rewrite all entries after you change the settings. To force the DS server to compress all entries, you must import the data from LDIF.

LDIF Import Settings

By default, the temporary directory used for scratch files is `opendj/import-tmp` . Use the **`import-ldif --tmpDirectory`** option to set this directory to a `tmpfs` file system, such as `/tmp` .

If you are certain your LDIF contains only valid entries with correct syntax, you can skip schema validation. Use the **`import-ldif --skipSchemaValidation`** option.

Database Cache Settings

IMPORTANT

IMPORTANT

By default, DS directory servers:

- Use shared cache for all JE database backends.

  The recommended setting is to leave the global property, `je-backend-shared-cache-enabled`, set to `true`.

  If you have more than one JE database backend, *before* you change this setting to `false`, you must set either `db-cache-percent` or `db-cache-size` appropriately for each JE backend. By default, `db-cache-percent` is 50% for each backend. If you have multiple backends, including backends created with setup profiles, the default settings can prevent the server from starting if you first disable the shared cache.

- Cache JE database internal and leaf notes to achieve best performance.

  The recommended setting is to leave this advanced property, `db-cache-mode`, set to `cache-ln`.

  In very large directory deployments, monitor the server to make sure internal nodes remain cached. For details, see Cache Internal Nodes.

If you require fine-grained control over JE backend cache settings, you can configure the amount of memory requested for database cache per database backend:

1. Configure `db-cache-percent` or `db-cache-size` for each JE backend.

   `db-cache-percent`
   : Percentage of JVM memory to allocate to the database cache for the backend.

     If the directory server has multiple database backends, the total percent of JVM heap used must remain less than 100 (percent), and must leave space for other uses.

     Default: 50 (percent)

   `db-cache-size`
   : JVM memory to allocate to the database cache.

     This is an alternative to `db-cache-percent`. If you set its value larger than 0, then it takes precedence over `db-cache-percent`.

     Default: 0 MB

2. Set the global property `je-backend-shared-cache-enabled:false`.
3. Restart the server for the changes to take effect.

Cache Internal Nodes

A JE backend is implemented as a B-tree data structure. A B-tree is made up of nodes that can have children. Nodes with children are called *internal nodes*. Nodes without children are called *leaf nodes*.

The directory stores data in key-value pairs. Internal nodes hold the keys, and can also hold small values. Leaf nodes hold the values. One internal node usually holds keys to values in many leaf nodes. A B-tree has many more leaf nodes than internal nodes.

To read a value by its key, the backend traverses all internal nodes on the branch from the B-tree root to the leaf node holding the value. The backend is more likely to access nodes the closer they are to the B-tree root. Internal nodes are accessed far more frequently than leaf nodes, and must remain cached in memory. In addition to the worker threads serving client application requests, cleaner threads working in the background also access internal nodes frequently. The performance impact of having to fetch frequently used internal nodes from disk can be severe.

When the database cache is full, the backend must begin evicting nodes from cache in order to load others. By default, the backend evicts leaf nodes even when the cache is not full. The backend is less likely to access a leaf node than an internal node, and leaf nodes might remain in the file system cache where they can be accessed quickly. If, however, the internal nodes do not all fit in cache, the backend eventually evicts even critical internal nodes.

Monitor the backend database environment to react if a backend evicts internal nodes, or performs critical evictions. The following example shows no internal node (IN) evictions, and no critical evictions:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN cn=backends,cn=monitor \
 "(|(ds-mon-db-cache-evict-internal-nodes-count=*)(ds-mon-je-
environment-nbytes-evicted-critical=*))" \
 ds-mon-db-cache-evict-internal-nodes-count \
 ds-mon-je-environment-nbytes-evicted-critical

dn: ds-cfg-backend-id=dsEvaluation,cn=backends,cn=monitor
ds-mon-db-cache-evict-internal-nodes-count: 0

dn: cn=raw JE database statistics,ds-cfg-backend-
```

```
id=dsEvaluation,cn=backends,cn=monitor
ds-mon-je-environment-nbytes-evicted-critical: 0
```

- If `ds-mon-db-cache-evict-internal-nodes-count` is greater than `0`, then the system has too little memory for all internal nodes to remain in DB cache.
- If `ds-mon-je-environment-nbytes-evicted-critical` is greater than `0`, then the DB worker threads are evicting data because the normal process of clearing cache using background threads is no longer sufficient.

Increase the DB cache size, and add more RAM to your system if necessary, until there are no internal node evictions, and no critical evictions. If adding RAM is not an option, increase the maximum heap size (`-Xmx`) to optimize RAM allocation.

Estimate Minimum DB Cache Size

When the DB cache is not large enough to hold all internal nodes, the performance impact can be severe. This section explains how to estimate the minimum DB cache size to hold all internal nodes.

The examples below reflect a directory server with a 10 million entry `dsEvaluation` backend. The backend holds Example.com entries that generated as described in Install DS for Evaluation with the additional setup option `--set ds-evaluation/generatedUsers:10,000,000`.

Base your own calculations on realistic sample data, with the same indexes that you use in production, and with data affected by realistic client application and replication loads. To generate your own sample data, start by reading Generate Test Data. To simulate load, use the tools described in Performance Tests. Even better, learn about real loads from analysis of production access logs, and build custom test clients that reflect the access patterns of your applications.

NOTE

After you import LDIF, the backend contains the minimum number of internal nodes required for the data. Over time as external applications update the directory server, the number of internal nodes grows.

A JE backend only appends to the database log for update operations, so many internal nodes in the database logs of a live system represent garbage that the backend eventually cleans up. Only the live internal nodes must be cached in memory. Over time, the increase in the number of internal nodes should track backend growth.

After loading the server for some time, stop the server. Use the `backendstat` command and JE `DbCacheSize` tool together to estimate the required DB cache size.

The following example uses the `backendstat` command to discover information about keys in the backend. Using a script or a spreadsheet on the output, calculate the total number of keys (sum of Total Keys, here: 73255315) and average key size (sum of Key Size/sum of Total Keys, here: 13). Use the results as input to the JE `DbCacheSize` tool:

```
# Stop the server before using backendstat:
$ stop-ds

$ backendstat list-raw-dbs --backendId dsEvaluation
Raw DB Name              ...   Total Keys  Keys Size  Values Size
Total Size
------------------------ ... ------------------------------------
----------
/compressed_schema/comp ...   50          50         772
822
/compressed_schema/comp ...   17          17         848
865
/dc=com,dc=example/aci. ...   1           1          3
4
/dc=com,dc=example/cn.c ...   10000165    139242471  47887210
187129681
/dc=com,dc=example/cn.c ...   858658      5106085    204936391
210042476
/dc=com,dc=example/dn2i ...   10000181    268892913  80001448
348894361
/dc=com,dc=example/ds-c ...   0           0          0
0
/dc=com,dc=example/ds-c ...   1           18         3
21
/dc=com,dc=example/ds-s ...   0           0          0
0
/dc=com,dc=example/ds-s ...   0           0          0
0
/dc=com,dc=example/entr ...   9988518     39954072   47871653
87825725
/dc=com,dc=example/give ...   8614        51691      20017387
20069078
/dc=com,dc=example/give ...   19652       97670      48312528
48410198
/dc=com,dc=example/id2c ...   8           26         14
40
/dc=com,dc=example/id2e ...   10000181    80001448   4989592300
5069593748
/dc=com,dc=example/json ...   4           74         10
84
```

```
/dc=com,dc=example/json ...   2           34         4
38
/dc=com,dc=example/mail ...   10000152    238891751  47887168
286778919
/dc=com,dc=example/mail ...   1222798     7336758    112365106
119701864
/dc=com,dc=example/memb ...   1           40         2
42
/dc=com,dc=example/obje ...   23          379        393
772
/dc=com,dc=example/refe ...   0           0          0
0
/dc=com,dc=example/sn.c ...   13457       92943      20027045
20119988
/dc=com,dc=example/sn.c ...   41585       219522     73713958
73933480
/dc=com,dc=example/stat ...   23          1153       22
1175
/dc=com,dc=example/tele ...   9989952     109889472  47873522
157762994
/dc=com,dc=example/tele ...   1111110     6543210    221282026
227825236
/dc=com,dc=example/uid. ...   10000152    118889928  47887168
166777096
/dc=com,dc=example/uniq ...   10          406        21
427

Total: 29

# Calculate sum of Total Keys, sum of Key Size, and average key
size:
$ java -cp /path/to/opendj/lib/opendj.jar
com.sleepycat.je.util.DbCacheSize \
 -records 73255315 -key 13
=== Environment Cache Overhead ===

3,158,773 minimum bytes

To account for JE daemon operation, record locks, HA network
connections, etc,
a larger amount is needed in practice.

=== Database Cache Size ===

   Number of Bytes  Description
```

```
   ---------------  -----------
     2,709,096,544  Internal nodes only

To get leaf node sizing specify -data

For further information see the DbCacheSize javadoc.
```

The resulting recommendation for DB cache size, 2,709,096,544 bytes in this case, is a minimum estimate. Round up when configuring backend settings for `db-cache-percent` or `db-cache-size`. If the system in this example has 8 GB available memory, use the default setting of `db-cache-percent: 50`. (50% * 8 GB = 4 GB, which is larger than the minimum estimate.)

Database Log File Settings

With default settings, if the database has more than 200 files on disk, then the JE backend must start closing one log file in order to open another. This has serious impact on performance when the file cache starts to thrash.

Having the JE backend open and close log files from time to time is okay. Changing the settings is only necessary if the JE backend has to open and close the files very frequently.

A JE backend stores data on disk in append-only log files. The maximum size of each log file is configurable. A JE backend keeps a configurable maximum number of log files open, caching file handles to the log files. The relevant JE backend settings are the following:

db-log-file-max
: Maximum size of a database log file.

  Default: 1 GB

db-log-filecache-size
: File handle cache size for database log files.

  Default: 200

With these defaults, if the size of the database reaches 200 GB on disk (1 GB x 200 files), the JE backend must close one log file to open another. To avoid this situation, increase `db-log-filecache-size` until the JE backend can cache file handles to all its log files. When changing the settings, make sure the maximum number of open files is sufficient.

Cache for Large Groups

DS servers implement an entry cache designed for a few large entries that are regularly updated or accessed, such as large static groups. An entry cache is used to keep such

groups in memory in a format that avoids the need to constantly read and deserialize the large entries.

When configuring an entry cache, take care to include only the entries that need to be cached. The memory devoted to the entry cache is not available for other purposes. Use the configuration properties `include-filter` and `exclude-filter` for this.

The following example adds a Soft Reference entry cache to hold entries that match the filter `(ou=Large Static Groups)`. A Soft Reference entry cache releases entries when the JVM runs low on memory. It does not have a maximum size setting. The number of entries cached is limited only by the `include-filter` and `exclude-filter` settings:

```
$ dsconfig \
 create-entry-cache \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --cache-name "Large Group Entry Cache" \
 --type soft-reference \
 --set cache-level:1 \
 --set include-filter:"(ou=Large Static Groups)" \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

The entry cache configuration takes effect when the entry cache is enabled.

Log Settings

Debug logs trace the internal workings of DS servers, and should be used sparingly. Be particularly careful when activating debug logging in high-performance deployments.

In general, leave other logs active for production environments to help troubleshoot any issues that arise.

For servers handling 100,000 operations per second or more, the access log can be a performance bottleneck. Each client request results in at least one access log message. Test whether disabling the access log improves performance in such cases.

The following command disables the JSON-based LDAP access logger:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname localhost \
```

```
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --publisher-name "Json File-Based Access Logger" \
 --set enabled:false \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

The following command disables the HTTP access logger:

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

Changelog Settings

By default, a replication server indexes change numbers for replicated user data. This allows legacy applications to get update notifications by change number, as described in Align Draft Change Numbers. Indexing change numbers requires additional CPU, disk accesses and storage, so it should not be used unless change number-based browsing is required.

Disable change number indexing if it is not needed. For details, see Disable Change Number Indexing.

Troubleshooting

Define the problem

To solve your problem, save time by clearly defining it first. A problem statement *compares the difference between observed behavior and expected behavior*:

- What exactly is the problem?

What is the behavior you expected?

What is the behavior you observed?

- How do you reproduce the problem?
- When did the problem begin?

  Under similar circumstances, when does the problem not occur?

- Is the problem permanent?

  Intermittent?

  Is it getting worse? Getting better? Staying the same?

Installation problems

Use the logs

Installation and upgrade procedures result in a log file tracing the operation. Look for this in the command output:

```
See file for a detailed log of this operation.
```

Antivirus interference

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.

JE initialization

When starting a directory server on a Linux system, make sure the server user can watch enough files. If the server user cannot watch enough files, you might see an error message in the server log such as this:

```
InitializationException: The database environment could not be
opened:
com.sleepycat.je.EnvironmentFailureException: (JE version)
/path/to/opendj/db/userData
or its sub-directories to WatchService.
UNEXPECTED_EXCEPTION: Unexpected internal Exception, may have side
effects.
Environment is invalid and must be closed.
```

File notification

A directory server backend database monitors file events. On Linux systems, backend databases use the inotify API for this purpose. The kernel tunable `fs.inotify.max_user_watches` indicates the maximum number of files a user can watch with the inotify API.

Make sure this tunable is set to at least 512K:

```
$ sysctl fs.inotify.max_user_watches

fs.inotify.max_user_watches = 524288
```

If this tunable is set lower than that, update the `/etc/sysctl.conf` file to change the setting permanently, and use the **`sysctl -p`** command to reload the settings:

```
$ echo fs.inotify.max_user_watches=524288 | sudo tee -a
/etc/sysctl.conf
[sudo] password for admin:

$ sudo sysctl -p
fs.inotify.max_user_watches = 524288
```

Forgotten superuser password

By default, DS servers store the entry for the directory superuser in an LDIF backend. Edit the file to reset the password:

1. Generate the encoded version of the new password:

   ```
   $ encode-password --storageScheme PBKDF2-HMAC-SHA256 --
   clearPassword password

   {PBKDF2-HMAC-SHA256}10<hash>
   ```

2. Stop the server while you edit the LDIF file for the backend:

   ```
   $ stop-ds
   ```

3. Replace the existing password with the encoded version.

   In the `db/rootUser/rootUser.ldif` file, carefully replace the `userPassword` value with the new, encoded password:

   ```
   dn: uid=admin
   ...
   uid: admin
   userPassword: <encoded-password>
   ```

   Trailing whitespace is significant in LDIF. *Take care not to add any trailing whitespace at the end of the line.*

4. Restart the server:

```
$ start-ds
```

5. Verify that you can use the directory superuser account with the new password:

```
$ status \
 --bindDn uid=admin \
 --bindPassword password \
 --hostname localhost \
 --port 4444 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --script-friendly
...
"isRunning" : true,
```

Debug logging

CAUTION

DS debug logging can generate a high volume of debug messages. Use debug logging very sparingly on production systems.

1. Create one or more debug targets.

   No debug targets are enabled by default:

```
$ dsconfig \
 list-debug-targets \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

```
Debug Target : enabled : debug-exceptions-only
-------------:---------:----------------------
```

A debug target specifies a fully qualified DS Java package, class, or method:

```
$ dsconfig \
 create-debug-target \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --type generic \
 --target-name org.opends.server.api \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

2. Enable the debug log, `opendj/logs/debug` :

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

The server immediately begins to write debug messages to the log file.

3. Read messages in the debug log file:

```
$ tail -f /path/to/opendj/logs/debug
```

4. Disable the debug log as soon as it is no longer required.

Lockdown Mode

Misconfiguration can put the DS server in a state where you must prevent users and applications from accessing the directory until you have fixed the problem.

DS servers support *lockdown mode* . Lockdown mode permits connections only on the loopback address, and permits only operations requested by superusers, such as `uid=admin` .

To put the DS server into lockdown mode, the server must be running. You cause the server to enter lockdown mode by starting a task. Notice that the modify operation is performed over the loopback address (accessing the DS server on the local host):

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: ds-task-id=Enter Lockdown Mode,cn=Scheduled Tasks,cn=tasks
objectClass: top
objectClass: ds-task
ds-task-id: Enter Lockdown Mode
ds-task-class-name: org.opends.server.tasks.EnterLockdownModeTask
EOF
```

The DS server logs a notice message in `logs/errors` when lockdown mode takes effect:

```
...msg=Lockdown task Enter Lockdown Mode finished execution
```

Client applications that request operations get a message concerning lockdown mode:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN "" \
 --searchScope base \
 "(objectclass=*)" \
 +

# The LDAP search request failed: 53 (Unwilling to Perform)
```

```
# Additional Information:  Rejecting the requested operation
because the server is in lockdown mode and will only accept
requests from root users over loopback connections
```

Leave lockdown mode by starting a task:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: ds-task-id=Leave Lockdown Mode,cn=Scheduled Tasks,cn=tasks
objectClass: top
objectClass: ds-task
ds-task-id: Leave Lockdown Mode
ds-task-class-name: org.opends.server.tasks.LeaveLockdownModeTask
EOF
```

The DS server logs a notice message when leaving lockdown mode:

```
...msg=Leave Lockdown task Leave Lockdown Mode finished execution
```

LDIF import

- By default, DS directory servers check that entries you import match the LDAP schema.

  You can temporarily bypass this check with the **`import-ldif --skipSchemaValidation`** option.

- By default, DS servers ensure that entries have only one structural object class.

  You can relax this behavior with the advanced global configuration property, `single-structural-objectclass-behavior`.

  This can be useful when importing data exported from Sun Directory Server.

  For example, warn when entries have more than one structural object class, rather than rejecting them:

  ```
  $ dsconfig \
   set-global-configuration-prop \
  ```

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

- By default, DS servers check syntax for several attribute types. Relax this behavior using the advanced global configuration property, `invalid-attribute-syntax-behavior`.
- Use the **`import-ldif -R rejectFile --countRejects`** options to log rejected entries and to return the number of rejected entries as the command's exit code.

Once you resolve the issues, reinstate the default behavior to avoid importing bad data.

Security problems

Incompatible Java versions

Due to a change in Java APIs, the same DS deployment ID generates different CA key pairs with Java 11 and Java 17 and later. When running the **`dskeymgr`** and **`setup`** commands, use the same Java environment everywhere in the deployment.

Using different Java versions is a problem if you use deployment ID-based CA certificates. Replication breaks, for example, when you use the **`setup`** command for a new server with a more recent version of Java than was used to set up existing servers. The error log includes a message such as the following:

```
...category=SYNC severity=ERROR msgID=119 msg=Directory server
DS(server_id)
encountered an unexpected error while connecting to replication
server host:port for domain "base_dn":
ValidatorException: PKIX path validation failed:
java.security.cert.CertPathValidatorException:
signature check failed
```

To work around the issue, follow these steps:

1. Update all DS servers to use the same Java version.

   Make sure you have a required Java environment installed on the system.

If your default Java environment is not appropriate, use one of the following solutions:

- Edit the `default.java-home` setting in the `opendj/config/java.properties` file.
- Set `OPENDJ_JAVA_HOME` to the path to the correct Java environment.
- Set `OPENDJ_JAVA_BIN` to the absolute path of the **`java`** command.

2. Export CA certificates generated with the different Java versions.

   a. Export the CA certificate from an old server:

```
$ keytool \
 -exportcert \
 -alias ca-cert \
 -keystore /path/to/old-server/config/keystore \
 -storepass:file /path/to/old-server/config/keystore.pin \
 -file java11-ca-cert.pem
```

   b. Export the CA certificate from a new server:

```
$ keytool \
 -exportcert \
 -alias ca-cert \
 -keystore /path/to/new-server/config/keystore \
 -storepass:file /path/to/new-server/config/keystore.pin \
 -file java17-ca-cert.pem
```

3. On *all* existing DS servers, import the *new* CA certificate:

```
$ keytool \
 -importcert \
 -trustcacerts \
 -alias alt-ca-cert \
 -keystore /path/to/old-server/config/keystore \
 -storepass:file /path/to/old-server/config/keystore.pin \
 -file java17-ca-cert.pem \
 -noprompt
```

4. On *all* new DS servers, import the *old* CA certificate:

```
$ keytool \
 -importcert \
 -trustcacerts \
 -alias alt-ca-cert \
 -keystore /path/to/new-server/config/keystore \
```

```
-storepass:file /path/to/new-server/config/keystore.pin \
-file java11-ca-cert.pem \
-noprompt
```

The servers reload their keystores dynamically and replication works as expected.

Certificate-based authentication

Replication uses TLS to protect directory data on the network. Misconfiguration can cause replicas to fail to connect due to handshake errors. This leads to repeated error log messages in the `replication` log file such as the following:

```
...msg=Replication server accepted a connection from address
 to local address address but the SSL handshake failed.
 This is probably benign, but may indicate a transient network
outage
 or a misconfigured client application connecting to this
replication server.
 The error was: Received fatal alert: certificate_unknown
```

You can collect debug trace messages to help determine the problem. To see the TLS debug messages, start the server with `javax.net.debug` set:

```
$ OPENDJ_JAVA_ARGS="-Djavax.net.debug=all" start-ds
```

The debug trace settings result in many, many messages. To resolve the problem, review the output of starting the server, looking in particular for handshake errors.

If the chain of trust for your PKI is broken somehow, consider renewing or replacing keys, as described in Key Management. Make sure that trusted CA certificates are configured as expected.

Compromised keys

How you handle the problem depends on which key was compromised:

- For keys generated by the server, or with a deployment key, see Retire secret keys.
- For a private key whose certificate was signed by a CA, contact the CA for help. The CA might choose to publish a certificate revocation list (CRL) that identifies the certificate of the compromised key.

  Replace the key pair that has the compromised private key.
- For a private key whose certificate was self-signed, replace the key pair that has the compromised private key.

Make sure the clients remove the compromised certificate from their truststores. They must replace the certificate of the compromised key with the new certificate.

Client problems

Use the logs

By default, DS servers record messages for LDAP client operations in the `logs/ldap-access.audit.json` log file.

▼ Show example log messages


```
[
{
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "<clientIp>",
    "port": 12345
  },
  "server": {
    "ip": "<clientIp>",
    "port": 1389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "CONNECT",
    "connId": 0
  },
  "transactionId": "0",
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "0",
    "elapsedTime": 0,
    "elapsedTimeUnits": "MILLISECONDS"
  },
  "timestamp": "<timestamp>",
  "_id": "<uuid>"
},
{
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "<clientIp>",
    "port": 12345
  },
  "server": {
```

```
    "ip": "<clientIp>",
    "port": 1389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "SEARCH",
    "connId": 0,
    "msgId": 1,
    "dn": "dc=example,dc=com",
    "scope": "sub",
    "filter": "(uid=bjensen)",
    "attrs": ["ALL"]
  },
  "transactionId": "0",
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "0",
    "elapsedTime": 9,
    "elapsedTimeUnits": "MILLISECONDS",
    "nentries": 1
  },
  "timestamp": "<timestamp>",
  "_id": "<uuid>"
},
{
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "<clientIp>",
    "port": 12345
  },
  "server": {
    "ip": "<clientIp>",
    "port": 1389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "UNBIND",
    "connId": 0,
    "msgId": 2
  },
  "transactionId": "0",
  "timestamp": "<timestamp>",
  "_id": "<uuid>"
},
{
```

```
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "<clientIp>",
    "port": 12345
  },
  "server": {
    "ip": "<clientIp>",
    "port": 1389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "DISCONNECT",
    "connId": 0
  },
  "transactionId": "0",
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "0",
    "elapsedTime": 0,
    "elapsedTimeUnits": "MILLISECONDS",
    "reason": "Client Unbind"
  },
  "timestamp": "<timestamp>",
  "_id": "<uuid>"
}
]
```

Each message specifies the operation performed, the client that requested the operation, and when it completed.

By default, the server does not log internal LDAP operations corresponding to HTTP requests. To match HTTP client operations to internal LDAP operations:

1. Prevent the server from suppressing log messages for internal operations.

   Set `suppress-internal-operations:false` on the LDAP access log publisher.
2. Match the `request/connId` field in the HTTP access log with the same field in the LDAP access log.

Client access

To help diagnose client errors due to access permissions, see Effective rights.

Simple paged results

For some versions of Linux, you see a message in the DS access logs such as the following:

```
The request control with Object Identifier (OID)
"1.2.840.113556.1.4.319"
cannot be used due to insufficient access rights
```

This message means clients are trying to use the simple paged results control without authenticating. By default, a global ACI allows only authenticated users to use the control.

To grant anonymous (unauthenticated) user access to the control, add a global ACI for anonymous use of the simple paged results control:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword "password" \
 --add global-aci:"(targetcontrol=\"SimplePagedResults\") \
 (version 3.0; acl \"Anonymous simple paged results access\";
allow(read) \
 userdn=\"ldap:///anyone\";)" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Replication problems

Replicas do not connect

If you set up servers with different deployment keys, they cannot share encrypted data. By default, they also cannot trust each other's secure connections. You may see messages like the following in the `logs/replication` log file:

```
msg=Replication server accepted a connection from /address:port
to local address /address:port but the SSL handshake failed.
```

Unless the servers use your own CA, make sure their keys are generated with the same deployment key/password. Either set up the servers again with the same deployment key, or see Replace Deployment Keys.

Temporary Delays

Replication can generally recover from conflicts and transient issues. Temporary delays are normal and expected while replicas converge, especially when the write load is heavy. This is a feature of eventual convergence, not a bug.

For more information, see Replication Delay (LDAP).

Use the Logs

Replication uses its own error log file, `logs/replication`. Error messages in the log file have `category=SYNC`.

The messages have the following form. The following example message is folded for readability:

```
...msg=Replication server accepted a connection from
10.10.0.10/10.10.0.10:52859
 to local address 0.0.0.0/0.0.0.0:8989 but the SSL handshake
failed.
 This is probably benign, but may indicate a transient network
outage
 or a misconfigured client application connecting to this
replication server.
 The error was: Remote host closed connection during handshake
```

Stale Data

DS servers maintain historical information to bring replicas up to date, and to resolve conflicts. To prevent historical information from growing without limit, servers purge historical information after a configurable delay (replication-purge-delay, default: 3 days). A replica can become irrevocably out of sync if you restore it from a backup that is older than the purge delay, or if you stop it for longer than the purge delay. If this happens, reinitialize the replica from a recent backup or from a server that is up to date.

Incorrect configuration

When replication is configured incorrectly, fixing the problem can involve adjustments on multiple servers. For example, adding or removing a bootstrap replication server means updating the `bootstrap-replication-server` settings in the synchronization provider configuration of other servers. (The settings can be hard-coded in the configuration, or read from the environment at startup time, as described in Property Value Substitution. In either case, changing them involves at least restarting the other servers.)

For details, see sections in Replication.

Support

Sometimes you cannot resolve a problem yourself, and must ask for help or technical support. In such cases, identify the problem and how you reproduce it, and the version where you see the problem:

```
$ status --offline --version

ForgeRock Directory Services 7.1.8
Build <datestamp>
```

Be prepared to provide the following additional information:

- The Java home set in `config/java.properties`.
- Access and error logs showing what the server was doing when the problem started occurring.
- A copy of the server configuration file, `config/config.ldif`, in use when the problem started occurring.
- Other relevant logs or output, such as those from client applications experiencing the problem.
- A description of the environment where the server is running, including system characteristics, hostnames, IP addresses, Java versions, storage characteristics, and network characteristics. This helps to understand the logs, and other information.
- The `.zip` file generated using the **`supportextract`** command.

  For an example showing how to use the command, see supportextract.