FORGEROCK®

Deployment Guide

/ Directory Services 6.5

Latest update: 6.5.6

Mark Craig

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to deploying ForgeRock® Directory Services.

Table of Contents

Preface

This guide focuses on how to use Directory Services software to build secure, high-performance, manageable directory services. It helps directory service architects design services that fit their needs, whether the directory service is small enough to be embedded in another application or large enough to be deployed in multiple regions worldwide.

This guide includes the following chapters:

- "*Understanding Directory Components*"

  This chapter characterizes the roles played by directory service components.

- "*Planning the Project*"

  This chapter provides an overview of how to plan your directory services deployment project.

- "*Creating a Comprehensive Deployment Plan*"

  This chapter details steps to follow when preparing a directory services deployment plan.

- "*Applying Deployment Patterns*"

  This chapter proposes appropriate patterns for specific deployments.

- "*Provisioning Systems*"

  This chapter covers hardware and software prerequisites for your deployment.

- "*Deploying for DevOps and SaaS*"

  This chapter focuses on use of DS software in DevOps and SaaS deployments.

- "*Deployment Checklists*"

  This chapter offers checklists for deploying a directory service.

Chapter 1

Understanding Directory Components

A directory service provides LDAP and HTTP access to distributed, shared directory data. A deployed directory service consists of one or more components. Each component plays a particular role in your directory service. Before you design your deployment, you need to be familiar with the roles that each component can play.

This chapter characterizes the roles played by directory service components.

The software components, shown in "Directory Service Components" include the following:

- *Directory servers*, which maintain and serve requests for directory data.

  In most deployments, directory servers use data *replication* to ensure that their data sets eventually converge everywhere. This documentation refers to a replicated directory server as a *replica*.

- *Directory proxy servers* that forward requests for directory data to directory servers, and return directory server responses to client applications.

- *Replication servers* that transmit data replication messages among replicas.

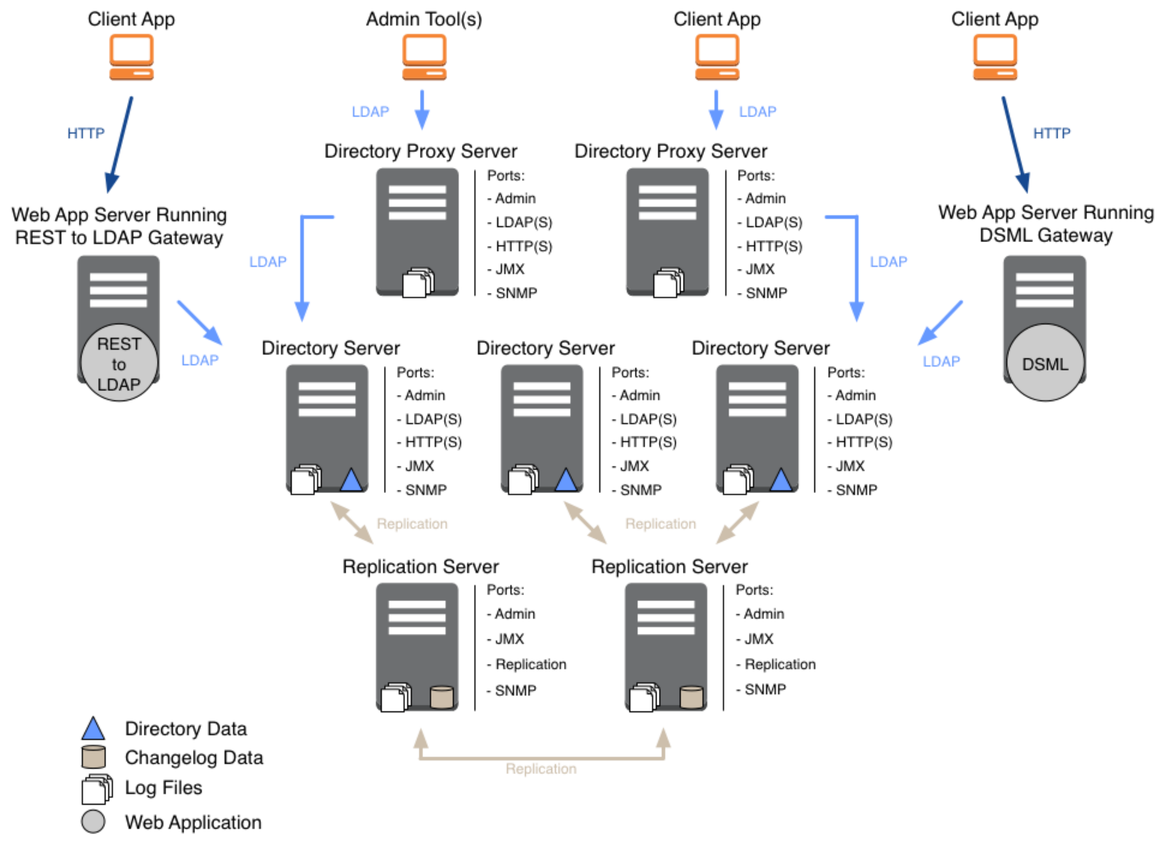
  You can configure a directory server to act as a replication server as well. A *standalone* replication server plays only the replication server role, brokering replication change messages. It does not store directory data.

- *DSML gateways* that intermediate between DSML client applications and an LDAP directory.

- *REST to LDAP gateways* that intermediate between RESTful HTTP client applications and LDAP directories.

- LDAP client tools and server administration tools for testing and managing servers.

Directory Service Components



Directory Servers

This section describes the roles and characteristics of directory servers.

Directory Server Roles

Directory servers provide access to their copy of the distributed directory database. A directory server usually functions as the repository of identities for users, applications, and things. They respond to requests from client applications directly or indirectly through directory proxy servers. This includes the following:

- LDAP requests for authentication, reads, and updates.

An LDAP client application authenticates with the directory server, and then performs one or more operations before either reauthenticating to reuse the connection or ending the session and closing the connection.

- HTTP read and update requests, often including credentials for authentication.

  An HTTP request translates to one or more internal LDAP requests.

- Administrative requests, such as requests to modify the server configuration or to perform a task such as backup or LDIF export.

- JMX and SNMP requests specifically for monitoring information.

In deployments with multiple *replicas*, directory servers replay replicated operations. Expect each replica to replay every successful update to any replica.

Directory Server Data

In addition to the libraries and tools delivered with the server distribution, a directory server is associated with the following persistent state information and local data:

User data

Directory servers store user data. The directory server stores the data in local storage, such as an internal disk or an attached disk array. The storage must keep pace with throughput for update operations.

The amount of user data depends entirely on the deployment, ranging from a few LDAP entries to more than a billion. The amount of user data grows or shrinks in deployment depending on the pattern of update operations.

The directory server stores user data in a backend database. For details, see "About Database Backends" in the *Administration Guide*.

Metadata for replication

A directory server can be a replica of other directory servers, meaning it can hold an eventually consistent copy of the data on the other replicas. To avoid an individual server becoming a single point of failure, almost all real-world deployments depend on replication.

When serving a request to update directory data, the directory server modifies its data and sends a request to a replication server. The replication server, described in "Replication Servers", ensures that all other replicas update their data to eventually reflect the current state of the data.

To tolerate network partitions, the directory service supports concurrent update operations on different replicas. Concurrent updates potentially cause conflicts, but directory servers can resolve most conflicts automatically. To resolve conflicts, a directory server stores historical metadata alongside user data, trading space for resilience. For details, see "About Replication" in the *Administration Guide*.

The directory server purges this historical metadata after a configurable interval. The volume of historical metadata depends on the total number of updates made to the directory service since the purge interval.

Server configuration

Each server has configuration data in its `config` directory. This includes the server configuration, mostly in LDIF format, LDAP schema definitions also in LDIF format, keys used to secure connections and perform encryption and decryption, and some additional data.

When installing a server, the **setup** command instantiates this configuration data from templates.

When upgrading a server, the **upgrade** command modifies this configuration data to apply any necessary changes.

Log files

The server writes to multiple log files by default, including error and access logs.

The server writes a message to the current access log for each operation. For high-volume directory services, log file storage must keep pace with the requests to record access to the service.

Log file retention and rotation policies prevent log file data from filling the disk. For details, see "Server Logs" in the *Administration Guide*. As a result of default retention policies, messages can eventually be lost unless you copy old files to another system for permanent storage.

Backup archives

When you export directory data to LDIF or create a backup archive, the directory server writes the files to the local file system. If you never purge or move these files, they can eventually fill the disk.

For details, see "Importing and Exporting Data" in the *Administration Guide* and "*Backing Up and Restoring Data*" in the *Administration Guide*.

Directory Server System Resources

When deciding how to deploy a directory server, think of it as a copy of the database. A large, high-performance, distributed database serving lots of applications obviously requires more system resources than a small database embedded in an application.

A directory server requires the following system resources:

- Sufficient RAM to cache frequently used data.

For best read performance, cache the entire directory data set in memory.

- Sufficient CPU to perform any required calculations.

  Authentication operations generally use more CPU than other operations. In particular, password storage schemes like PBKDF2 are designed to consume CPU resources. Calculations for transport layer security can use CPU as well, particularly if many client requests are over short-lived HTTPS connections.

- Sufficient fast disk access to serve client applications, to replay replication operations, and to log access and errors.

  The underlying disk subsystem must serve enough input/output operations per second (IOPS) to avoid becoming a bottleneck when performing these operations. A small database that serves few client operations and changes relatively infrequently requires fewer IOPS than a large database sustaining many updates and serving many clients.

  Plan additional capacity for any backup archives and LDIF export files stored on local partitions.

- Sufficiently fast network access to serve client applications and relay replication traffic.

  When considering network requirements, keep the following points in mind:

  - Each LDAP search request can return multiple response messages.

  - Each request to update directory data results in corresponding replication traffic. The operation must be communicated to replication servers and replayed on each other directory server.

  - Once established, and unlike most HTTP connections, LDAP connections remain open until the client closes the connection, or until the server idles the connection. This is particularly true for applications using persistent searches, which by design are intended to be permanent.

Replication Servers

This section describes the roles and characteristics of standalone replication servers. If you configure a directory server to play the role of a replication server as well, then the directory server also has these roles and characteristics.

Replication Server Roles

Replication servers provide the following services:

- Receive and transmit change messages between replicas.

  Each replica is connected to one replication server at a time. A single standalone replication server can serve 10 or more replicas.

- Maintain information about all other replication servers and directory servers in the deployment that replicate the same data.

  Change messages travel from a connected directory server to the replication server. The replication server transmits the message to connected replicas. If there are other replication servers, the replication server transmits the message to the other replication servers, which in turn transmit the message to their connected replicas. This hub-and-spoke communication model means directory services can be composed of many individual servers.

- Respond to administrative requests.

- Respond to HTTP, JMX, LDAP, and SNMP requests for monitoring information.

In all deployments using replication, the replication service provides the foundation of directory service availability. This is as important to the directory service as a naming service is for a network. When deploying replicated directory services, start by installing the replication service.

To avoid a single point of failure, install two or more replication servers in each location.

Replication Server Data

In addition to the libraries and tools delivered with the server distribution, a replication server is associated with the following persistent state information and local data:

Change data

When serving a request to update directory data, a directory server, described in "Directory Servers", modifies its data and sends a request to a replication server. The replication server ensures that all other replicas update their data to eventually reflect the current state of the data.

The replication protocol is proprietary. Replication servers expose a public record of changes in a change log, allowing other applications to keep up to date with changes to user data. This change data is stored in change log files. For details, see "Change Notification For Your Applications" in the *Administration Guide*.

The replication server purges this historical metadata after a configurable interval. The volume of historical metadata depends on the updates made to the directory service since the purge interval.

Server configuration

Each server has configuration data in its `config` directory. This includes the server configuration, mostly in LDIF format, LDAP schema definitions also in LDIF format, keys used to secure connections and perform encryption and decryption, and some additional data.

When installing a server, the **setup** command instantiates this configuration data from templates.

When upgrading a server, the **upgrade** command modifies this configuration data to apply any necessary changes.

Log files

The server writes to multiple log files by default, including error and access logs.

Log file retention and rotation policies prevent log file data from filling the disk. For details, see "Server Logs" in the *Administration Guide*. This means, however, that messages are eventually lost unless you move old files to another system for permanent storage.

Replication Server System Resources

When deploying a replication server, keep its foundational role in mind. Directory servers communicate with other replicas through replication servers. Directory proxy servers rely on replication servers to find directory servers.

A replication server requires the following system resources:

- Sufficient fast disk access to log and read change messages, and to update access and error logs.

  The underlying disk subsystem must serve enough IOPS to avoid becoming a bottleneck when performing these operations.

- Sufficiently fast network access to receive and transmit change messages for multiple replicas and for each other replication server.

Directory Proxy Servers

This section describes the roles and characteristics of directory proxy servers.

Directory Proxy Roles

Directory proxy servers provide the following services:

- Balance load of requests to LDAP directory servers.
- Receive and transmit LDAP client requests to LDAP directory servers.
- Receive and transmit LDAP directory server responses to LDAP client applications.
- Respond to administrative requests.
- Respond to HTTP, JMX, LDAP, and SNMP requests for monitoring information.

A directory proxy server can hide the underlying directory service architecture from client applications, enabling you to build a single point of directory service access.

A directory proxy server can discover directory servers through a replication server. This capability relies, however, on the replication server configuration. If you use the proxy server with third-party directory service components, then you must manually maintain the network locations for directory servers.

A directory proxy server provides LDAP access to remote LDAP directory servers. If you want to provide HTTP access to remote LDAP directory servers, use the REST to LDAP gateway instead. For details, see "REST to LDAP Gateway".

Directory Proxy Data

In addition to the libraries and tools delivered with the server distribution, a directory proxy server is associated with the following persistent state information and local data:

Server configuration

Each server has configuration data in its `config` directory. This includes the server configuration, mostly in LDIF format, LDAP schema definitions also in LDIF format, keys used to secure connections and perform encryption and decryption, and some additional data.

When installing a server, the **setup** command instantiates this configuration data from templates.

When upgrading a server, the **upgrade** command modifies this configuration data to apply any necessary changes.

Log files

The server writes to multiple log files by default, including error and access logs.

Log file retention and rotation policies prevent log file data from filling the disk. For details, see "Server Logs" in the *Administration Guide*. This means, however, that messages are eventually lost unless you move old files to another system for permanent storage.

Directory Proxy System Resources

In order to route requests appropriately, a directory proxy server must decode incoming requests and encode ongoing requests. It must also decode and encode incoming and outgoing responses. When deploying a directory proxy server, keep this decoding and encoding in mind, because it explains why you might need as many proxy servers as directory servers.

A directory proxy server requires the following system resources:

- Sufficient fast disk access to update access and error logs.

  The underlying disk subsystem must serve enough IOPS to avoid becoming a bottleneck when performing these operations.

- Sufficiently fast network access to receive and transmit client requests and server responses.

- Sufficient CPU to perform any required calculations.

Request and response decoding and encoding consume CPU resources.

- Sufficient RAM to maintain active connections.

Client and Server Tools

When you install the files for a server component, those files include tools for setting up, upgrading, and configuring and maintaining the server and administrative tasks. The files also include LDAP command-line tools for sending LDAP requests and measuring directory service performance.

For details, see "Server Command-Line Tools" in the *Administration Guide*.

DSML Gateway

This section describes the roles and characteristics of the standalone DSML gateway web application.

You can install this component independently of directory services. For details, see "*Installing the DSML Gateway*" in the *Installation Guide*,

DSML Gateway Role

DSML gateways provide the following services:

- Receive HTTP DSML requests from client applications, and transmit them as LDAP requests to a directory service.
- Receive LDAP responses from a directory service, and transmit them as HTTP DSML responses to client applications.

A DSML gateway runs in a Java web application server. It is limited to one host:port combination for the LDAP directory service.

DSML Gateway Data

A DSML gateway maintains only its own service configuration, recorded in the web application `WEB-INF/web.xml` file. It depends on the host web application server for other services, such as logging.

DSML Gateway System Resources

A DSML gateway requires the following system resources:

- Sufficiently fast network access to receive and transmit client requests and server responses.
- Sufficient CPU to perform any required calculations.

Request and response decoding, encoding, and transformation all consume CPU resources.

Calculations to secure network connections also consume CPU resources.

- Sufficient RAM to maintain active connections.

REST to LDAP Gateway

This section describes the roles and characteristics of the standalone REST to LDAP gateway web application. REST refers to the representational state transfer architectural style. RESTful requests use the HTTP protocol.

You can install this component independently of directory services. For details, see "*Installing the REST to LDAP Gateway*" in the *Installation Guide*,

REST to LDAP Gateway Roles

REST to LDAP gateways provide the following services:

- Receive HTTP requests from client applications, and transmit them as LDAP requests to a directory service.
- Receive LDAP responses from a directory service, and transmit them as HTTP responses to client applications.

A REST to LDAP gateway runs in a Java web application server. It can be configured to contact multiple LDAP directory servers.

One RESTful HTTP request can generate multiple LDAP requests. This is particularly true if the REST to LDAP mapping configuration includes references to resolve before returning response entries. For example, an LDAP user entry can have a `manager` attribute that holds the DN of the user's manager's entry. Rather than return an LDAP-specific DN in the REST response, the REST to LDAP mapping is configured to return the manager's name in the response. As a result, every time a user's manager is returned in the response, the gateway must make a request for the user's LDAP information and another request for the user's manager's name.

REST to LDAP Gateway Data

A REST to LDAP gateway maintains only its own service configuration, recorded in files as described in "*REST to LDAP Configuration*" in the *Reference*. It depends on the host web application server for other services, such as logging.

REST to LDAP Gateway System Resources

A REST to LDAP gateway requires the following system resources:

- Sufficiently fast network access to receive and transmit client requests and server responses.
- Sufficient CPU to perform any required calculations.

  Request and response decoding, encoding, and transformation all consume CPU resources.

  Calculations to secure network connections also consume CPU resources.
- Sufficient RAM to maintain active connections.

Chapter 2
Planning the Project

This chapter provides an overview of how to plan your directory services deployment project.

For a detailed discussion of how to design your service, see "*Creating a Comprehensive Deployment Plan*".

Importance of Needs Assessment

Needs assessment is prerequisite to developing a comprehensive deployment plan. An accurate needs assessment is critical to ensuring that your directory services implementation meets your business needs and objectives.

As part of the needs assessment, make sure you answer the following questions:

What are your business objectives?

Clarify and quantify your business goals for directory services.

Why do you want to deploy directory services?

Consider at least the following list when answering this question:

- Is this a greenfield deployment?
- Do you need to transition an existing deployment to the cloud?
- Do you need to scale existing deployment for more users, devices, or things?

If you have an existing deployment, how do you upgrade?

Consider at least the following list when answering this question:

- Do you require a graceful upgrade?
- What obsolete components need a graceful transition?

  What should their replacements be?
- What are the costs related to the change?

How can you save cost by making the transition?

Define objectives based on your needs assessment. State your objective so that all stakeholders agree on the same goals and business objectives.

Importance of Deployment Planning

Deployment planning is critical to ensuring that your directory services are properly implemented within the time frame determined by your requirements. The more thoroughly you plan your deployment, the more solid your configuration will be, and you will meet timelines and milestones while staying within budget.

A deployment plan defines the goals, scope, roles, and responsibilities of key stakeholders, architecture, implementation, and testing of your DS deployment. A good plan ensures that a smooth transition to a new product or service is configured and all possible contingencies are addressed to quickly troubleshoot and solve any issue that may occur during the deployment process. The deployment plan also defines a training schedule for your employees, procedural maintenance plans, and a service plan to support your directory services.

Deployment Planning Considerations

When planning a deployment, you must consider some important questions regarding your system:

- **What key applications does your system serve?** Understand how key client applications will use your directory service and what they require. Based on this understanding, you can match service level objectives (SLOs) to operational requirements. This ensures that you focus on what is critical to your primary customers.
- **What directory data does your system serve?** Directory data can follow standard schema and be shared by many applications. Alternatively, it can be dedicated to a single application such as AM CTS or IDM repository. Key applications can impose how they access directory data, or the directory data definition can be your decision.

  In addition, know where you will obtain production data, and in what format you will obtain it. You might need to maintain synchronization between your directory service and existing data services.
- **What are your SLOs?** In light of what you know about key and other applications, determine your SLOs. An SLO is a target for a directory service level that you can measure quantitatively.

  What objectives will you set for your service? How will you measure the following?

  - Availability
  - Response times

  - Throughput
  - Support response

- **What are your availability requirements?** DS services are designed to run continuously, without interruption even during upgrade. Providing a highly available service of course comes with operational complexities and costs.

  If your deployment must be highly available, take care in your planning phase to avoid single points of failure. You will need to budget for redundancy in all cases, and good operational policies, procedures, and training to avoid downtime as much as possible.

  If your deployment does not require true high availability, however, you will benefit from taking this into account during the planning stages of your deployment as well. You may be able to find significant cost savings as a trade for lower availability.

- **What are your security requirements?** DS services build in security in depth as described in the Security Guide.

  Understand the specific requirements of your deployment in order to use only the security features you really need. If you have evaluated DS software by setting up servers in the default security mode, carefully consider what you must configure explicitly after using the `setup --productionMode` option.

- **Are all stakeholders engaged starting in the planning phase?** This effort includes but is not limited to delivery resources, such as project managers, architects, designers, implementers, testers, and service resources, such as service managers, production transition managers, security, support, and sustaining personnel. Input from all stakeholders ensures all viewpoints are considered at project inception, rather than downstream, when it may be too late.

Deployment Planning Steps

The following sections summarize deployment planning steps for each phase.

Project Initiation

The project initiation phase begins by defining the overall scope and requirements of the deployment. Plan the following items:

- Determine the scope, roles and responsibilities of key stakeholders and resources required for the deployment.
- Determine critical path planning including any dependencies and their assigned expectations.
- Run a pilot to test the functionality and features of AM and uncover any possible issues early in the process.

- Determine training for administrators of the environment and training for developers, if needed.

Design

The design phase involves defining the deployment architecture. Plan the following items:

- Determine the use of products, map requirements to features, and ensure the architecture meets the functional requirements.
- Ensure that the architecture is designed for ease of management and scale. TCO is directly proportional to the complexity of the deployment.
- Define the directory data model.
- Determine how client applications will access directory data, and what data they have access to.
- Determine which, if any, custom DS server plugins must be developed. Derive specifications and project plans for each plugin.
- Determine the replication configuration.
- Define backup and recovery procedures, including how to recover sets of replicated servers should disaster occur.
- Define monitoring and audit procedures, and how the directory service integrates with your tools.
- Determine how to harden DS servers for a secure deployment.
- Define the change management process for configurations and custom plugins.
- Define the test criteria to validate that the service meets your objectives.
- Define the operations required to maintain and support the running service.
- Define how you will roll out the service into production.
- Determine how many of each DS server type to deploy in order to meet SLOs In addition, define the systems where each of the servers will run.

Implementation

The implementation phase involves deploying directory services. Plan the following items:

- Provision the DS servers.
- Maintain a record and history of the deployment for consistency across the project.
- Monitor and maintain the running service.

Automation, Continuous Integration, and Testing

The automation and continuous integration phase involves using tools for testing. Plan the following items:

- Use a continuous integration server, such as Jenkins, to ensure that changes have the expected impact, and no change causes any regressions.
- Ensure your custom plugins follow the same continuous integration process.
- Test all functionality to deliver the solution without any failures. Ensure that all customizations and configurations are covered in the test plan.
- Non-functionally tests failover and disaster recovery procedures. Run load testing to determine the demand of the system and measure its responses. During this phase, anticipate peak load conditions.

Supportability

The supportability phase involves creating the runbook for system administrators and operators. This includes procedures for backup and restore operations, debugging, change control, and other processes.

If you have a ForgeRock Support contract, it ensures everything is in place prior to your deployment.

Chapter 3
Creating a Comprehensive Deployment Plan

This chapter details steps to follow when preparing a directory services deployment plan.

Organizing Deployment Team Training

Training provides a common understanding, vocabulary, and basic skills for those working together on the project. Depending on previous experience with access management and with DS software, both internal teams and project partners might need training.

The type of training team members need depends on their involvement in the project:

- All team members should take at least some training that provides an overview of DS software. This helps to ensure a common understanding and vocabulary for those working on the project.
- Team members planning the deployment should take an DS training before finalizing their plans, and ideally before starting to plan the deployment.

  DS training pays for itself as it helps you to make the right initial choices to deploy more quickly and successfully.
- Team members involved in designing and developing DS client applications or custom plugins should take training in DS development in order to help them make the right choices.
- Team members who have already had been trained in the past might need to refresh their knowledge if your project deploys newer or significantly changed features, or if they have not worked with DS software for some time.

ForgeRock University regularly offers training courses for DS topics. For a current list of available courses, see the ForgeRock web site.

When you have determined who needs training and the timing of the training during the project, prepare a training schedule based on team member and course availability. Include the scheduled training plans in your deployment project plan.

ForgeRock also offers an accreditation program for partners, including an in-depth assessment of business and technical skills for each ForgeRock product. This program is open to the partner community and ensures that best practices are followed during the design and deployment phases.

Developing Custom Server Plugins

DS servers provide a Java plugin API that allows you to extend and customize server processing at a variety of plugin points. This keeps the core server processing focused on directory logic, and loosely coupled with other operations.

When you create your own custom plugin, be aware that you must at a minimum recompile and potentially update your plugin code for every DS server update. The plugin API has interface stability: Evolving. A plugin built with one version of a server is not guaranteed to run or even to compile with a subsequent version. Only create your own custom plugin when you require functionality that the server cannot be configured to provide. The best practice is to deploy DS servers with a minimum of custom plugins.

For more information about plugins, including instructions to build and use the example server plugin delivered with DS servers, see "*Writing a Server Plugin*" in the *Developer's Guide*.

Although some custom plugins involve little development work, they can require additional scheduling and coordination. The more you customize, the more important it is to test your deployment thoroughly before going into production. Consider each custom plugin as sub-project with its own acceptance criteria. Prepare separate plans for unit testing, automation, and continuous integration of each custom plugin. See "Planning Tests" for details.

When you have prepared plans for each custom plugin sub-project, you must account for those plans in your overall deployment project plan.

Piloting the Implementation

Unless you are planning a maintenance upgrade, consider starting with a pilot implementation, which is a long-term project that is aligned with your specific requirements.

A pilot shows that you can achieve your goals with DS software plus whatever custom plugins and companion software you expect to use. The idea is to demonstrate feasibility by focusing on solving key use cases with minimal expense, but without ignoring real-world constraints. The aim is to fail fast before you have too much invested so that you can resolve any issues that threaten the deployment.

Do not expect the pilot to become the first version of your deployment. Instead, build the pilot as something you can afford to change easily, and to throw away and start over if necessary.

The cost of a pilot should remain low compared to overall project cost. Unless your concern is primarily the scalability of your deployment, you run the pilot on a much smaller scale than the full deployment. Scale back on anything not necessary to validating a key use case.

Smaller scale does not necessarily mean a single-server deployment, though. If you expect your deployment to be highly available, for example, one of your key use cases should be continued smooth operation when part of your deployment becomes unavailable.

The pilot is a chance to try and test features and services before finalizing your plans for deployment. The pilot should come early in your deployment plan, leaving appropriate time to adapt your plans based on the pilot results. Before you can schedule the pilot, team members might need training. You might require prototype versions of functional customizations.

Plan the pilot around the key use cases that you must validate. Make sure to plan the pilot review with stakeholders. You might need to iteratively review pilot results as some stakeholders refine their key use cases based on observations.

Creating the Directory Data Model

Before you start defining how to store and access directory data, you must know what data you want to store, and how client applications use the data. This section starts with the assumption that you have or can generate representative data samples for planning purposes, and that you can produce representative client traffic for testing.

When defining the directory information tree (DIT) and data model for your service, answer the following questions:

- What additional schema definitions does your directory data require?

  See "Defining LDAP Schema Extensions".

- What are the appropriate base DNs and branches for your DIT?

  See "Planning the DIT".

- How will applications access the directory service? Over LDAP? Over HTTP?

  See "Preparing Data Views".

- Will a single team manage the directory service and the data? Will directory data management be a shared task, delegated to multiple administrators?

  See "Managing Data".

- What groups will be defined in your directory service?

  See "Using Groups".

- What sort of data will be shared across many directory entries? Should you define virtual or collective attributes to share this data?

  See "Using Shared Data".

- How should you cache data for appropriate performance?

  See "Caching Data".

- How will identities be managed in your deployment?

  See "Managing Identities".

Defining LDAP Schema Extensions

As described in "*Managing Schema*" in the *Administration Guide*, DS servers ship with many standard LDAP schema definitions. In addition, you can update LDAP schema definitions while the server is online.

This does not mean, however, that you can avoid schema updates for your deployment. Instead, unless the data for your deployment requires only standard definitions, you must add LDAP schema definitions before importing your data.

Follow these steps to prepare the schema definitions to add:

1. If your data comes from another LDAP directory service, translate the schema definitions used by the data from the existing directory service. Use them to start an LDIF modification list of planned schema updates, as described in "Updating Directory Schema" in the *Administration Guide*.

   The schema definitions might not be stored in the same format as DS definitions. Translating from existing definitions should be easier than creating new ones, however.

   As long as the existing directory service performs schema checking on update, the directory data you reuse already conforms to those definitions. You must apply them to preserve data integrity.

2. If your data comes from applications that define their own LDAP schema, add those definitions to your list of planned schema updates.

3. Match as much of your data as possible to the standard LDAP schema definitions listed in the LDAP Schema Reference.

4. Define new LDAP schema definitions for data that does not fit existing definitions. This is described in "About Directory Schema" in the *Administration Guide*, and "Updating Directory Schema" in the *Administration Guide*.

   Add these new definitions to your list.

   Avoid any temptation to modify or misuse standard definitions, as doing so can break interoperability.

Once your schema modifications are ready, use comments to document your choices in the source LDIF. Keep the file under source control. Apply a change control process to avoid breaking schema definitions in the future.

Perhaps you can request object identifiers (OIDs) for new schema definitions from an OID manager in your organization. If not, either take charge of OID assignment, or else find an owner who takes charge. OIDs must remain globally unique, and must not be reused.

Planning the DIT

When defining the base DNs and hierarchical structure of the DIT, keep the following points in mind:

- For ease of use, employ short, memorable base DNs with RDNs using well-known attributes.

  For example, you can build base DNs that correspond to domain names from domain component (`dc`) RDNs. The sample data for Example.com uses `dc=example,dc=com`.

  Well-known attributes used in base DNs include the following:

  `c`: country, a two-letter ISO 3166 country code
  `dc`: component of a DNS domain name
  `l`: locality
  `o`: organization
  `ou`: organizational unit
  `st`: state or province name

- For base DNs and hierarchical structures, depend on properties of the data that do not change.

  For example, the sample data places all user entries under `ou=People,dc=example,dc=com`. There is no need to move a user account when the user changes status, role in the organization, location, or any other property of their account.

- Introduce hierarchy in order to group similar entries.

  As an example of grouping similar entries, the following branches separate apps, devices, user accounts, and LDAP group entries:

  `ou=Apps,dc=example,dc=com`
  `ou=Devices,dc=example,dc=com`
  `ou=Groups,dc=example,dc=com`
  `ou=People,dc=example,dc=com`

  In this example, client application accounts belong under `ou=Apps`. A user account under `ou=People` for a device owner or subscriber can have an attribute referencing devices under `ou=Devices`. Device entries can reference their `owner` in `ou=People`. Group entries can include members from any branch. Their members' entries would reference the groups with `isMemberOf`.

- Otherwise, use hierarchy only as required for specific features. Such features include the following:

  Access control
  Data distribution
  Delegated administration
  Replication
  Subentries

  Use delegated administration when multiple administrators share the directory service. Each has access to manage a portion of the directory service or the directory data. By default, ACIs and subentries apply to the branch beneath their entry or parent. If a delegated administrator must be able to add or modify such operational data, the DIT should prevent the delegated administrator from affecting a wider scope than you intend to delegate.

As described in "About Replication" in the *Administration Guide*, the primary unit of replication is the suffix, which is specified by a base DN. If your replication configuration will require splitting a base DN into separate suffixes, create branches in your DIT for that purpose. For example use cases, read "*Applying Deployment Patterns*".

Once you have defined your DIT, arrange the directory data you import to follow its structure.

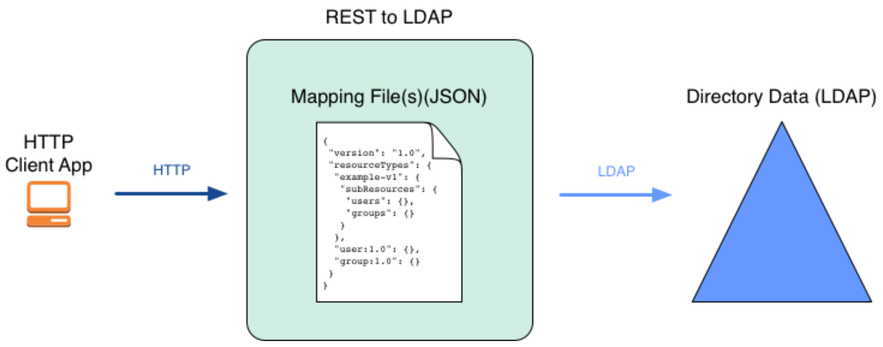
Preparing Data Views

If client applications only use LDAP to access the directory service, you have few choices to make when configuring connection handlers. The choices involve how to secure connections, and whether to limit access by client host name or address mask. The client applications all have the same LDAP view of the directory data.

This is true for DSML applications as well if you use the DSML gateway.

If client applications use RESTful HTTP APIs to access the directory service, then you have the same choices as for LDAP. In addition, you must define how the HTTP JSON resources map to LDAP entries.

Mapping JSON Resources to LDAP Entries



```
{
 "version": "1.0",
 "resourceTypes": {
  "example-v1": {
   "subResources": {
    "users": {},
    "groups": {}
   }
  },
  "user:1.0": {},
  "group:1.0": {}
 }
}
```

As shown above, you can define multiple versioned APIs providing alternative views of the underlying LDAP data if required for your deployment. The basic sample API that ships with DS servers is likely not sufficient. For details, see "*REST to LDAP Configuration*" in the *Reference*.

Managing Data

In a shared or high-scale directory service, service management—installation and configuration, backup, and recovery—may be the responsibility of only a few specialists. These tasks may be carefully scripted. "Directory Service Administration" in the *Security Guide* lists service management tasks.

Directory data management is, however, often a task shared by multiple users. "Directory Data Administration" in the *Security Guide* lists tasks that these user administrators perform. Many of these tasks may be performed manually. In addition, users may be responsible for profile data in their own entry, including passwords, for example. As described in "Planning the DIT", you can arrange the DIT hierarchically to make it easier to scope control of administrative access.

Your plan must define who should have what access to which data, and list the privileges and access controls to grant such access. Read "*Securing Directory Administration*" in the *Security Guide* to review the alternatives.

Using Groups

As described in "*Working With Groups of Entries*" in the *Developer's Guide*, DS directory servers offer static, dynamic, and virtual static group implementations:

- Static groups enumerate each member. The size of a static group entry can grow very large in a high-scale directory.
- Dynamic groups identify members by an LDAP URL, meaning that there is no static list of members.

  Instead of maintaining a static list of each member, members are those whose entries match a search filter. An entry could become a member of a dynamic group when one of its attributes changes.
- Virtual static groups are like dynamic groups, but the server can be configured to have them return a list of members when read.

Consider your data and client applications. Use dynamic or virtual static groups whenever possible. When you cannot use dynamic or virtual static groups, use static groups and consider caching them in memory as described in "Caching Large, Frequently Used Entries" in the *Administration Guide*.

Using Shared Data

As described in "*Working With Virtual and Collective Attributes*" in the *Developer's Guide*, DS servers support virtual and collective attributes that let entries share attribute values. Sharing attribute values where it makes sense can significantly reduce data duplication, saving space and avoiding maintenance updates.

Consider your directory data. You can use virtual or collective attributes to replace attributes that repeat on many entries and can remain read-only on those entries. Familiar use cases include postal addresses that are the same for everyone in a given location, and class of service properties that depend on a service level attribute.

Caching Data

A directory server is an object-oriented database. It will therefore exhibit best performance when its data is cached in memory. This is as true for large static groups mentioned in "Using Groups" as it is for all directory data.

A disadvantage of caching all data is that systems with enough RAM are more expensive. Furthermore, the JVM can grow so large that garbage collection and heap dumps become unwieldy. Consider the suggestions in "Database Cache Settings" in the *Administration Guide*, testing the results for your data when planning your deployment. Decide how to configure cache sizes based on your observations.

Managing Identities

DS servers have the following features that make them well-suited to serve identity data:

- LDAP entries provide a natural model for identity profiles and accounts.

  LDAP entries associate a unique name with a flat, extensible set of profile attributes such as credentials, location or contact information, descriptions, and more. LDAP schemas define what entries can contain, and are themselves extensible at runtime.

  Because they are defined and accessible in standard ways, and because fine-grained access controls can protect all attributes of each entry, the profiles can be shared by all network participants as the single source of identity information.

  Profile names need not be identified by LDAP DNs. For HTTP access, DS servers offer several ways to map to a profile, including mapping an HTTP user name to an LDAP name, or using an OAuth 2.0 access token instead. For devices and applications, DS servers can also map public key certificates to profiles.

- Directory services are optimized to support common authentication mechanisms.

  LDAP entries easily store and retrieve credentials, keys, PKI metadata, and more. Where passwords are used, directory services support multiple secure and legacy password storage schemes. You can also configure directory servers to upgrade password storage when users authenticate.

  Each individual server can process thousands of authentication requests per second.

  ForgeRock® Access Management integrates directory authentication into full access management services, including making directory authentication part of a flow that potentially involves multiple authentication steps.

- Directory services support user self-service operations and administrator intervention.

  Directory services let you protect accounts automatically or manually by locking accounts after repeated authentication failure, expiring old passwords, and tracking authentication times to distinguish active and inactive accounts. Directory services can then notify applications and users about account-related events, such as account lockout, password expiration, and other events.

Users can be granted access to update their own profiles and change their passwords securely. If necessary, administrators can also be granted access to update profiles and to reset passwords.

ForgeRock Identity Management integrates directory account management features into full identity management services.

Further Reading on Managing Identities

Topics	**References**
Account Management	"*Implementing Account Lockout and Notification*" in the *Administration Guide* "Search: Listing Active Accounts" in the *Developer's Guide*
Authentication	"*Securing Authentication*" in the *Security Guide* "Authenticating To the Directory Server" in the *Developer's Guide* "Authenticating Client Applications With a Certificate" in the *Developer's Guide* "*Configuring Pass-Through Authentication*" in the *Administration Guide* "Authenticating Over REST" in the *Developer's Guide*
Authorization	"To Set Up HTTP Authorization" in the *Administration Guide* "Configuring Proxied Authorization" in the *Developer's Guide*
Password Management	"*Configuring Password Policy*" in the *Administration Guide* "Changing Passwords" in the *Developer's Guide* "Using the Modify Password and Reset Password Actions" in the *Developer's Guide*

Creating a Directory Access Model

When designing the access model for your deployment, consider the topics in this section.

Separation of Duties (SoD)

The fewer restrictions you place on an administrative account, the greater the danger the account will be misused.

As described in "About the Roles Directory Administrators Play" in the *Security Guide*, you can avoid using directory superuser accounts for most operations. Instead, limit administrator privileges and access to grant only what their roles require. The first high-level distinction to make is between operational staff who manage the service, and those users who manage directory data. Read the section cited for fine-grained distinctions.

When your deployment involves delegated administration, it is particularly important to grant only required access to the delegates. This is easier if your DIT supports appropriate access scopes by default, as described in "Planning the DIT".

Immutable Versus Mutable Configuration

An immutable configuration does not change at runtime. A mutable configuration does change at runtime.

With an immutable configuration, you maintain the server configuration as an artifact under source control, and manage changes by applying the same process you use for other source code. This approach helps prevent surprises in production configurations. If properly applied, there is little risk of rolling out an untested change.

With a mutable configuration, operational staff have more flexibility to make changes. This approach requires even more careful change management for test and production systems.

DS server configurations can be immutable, except for the portion devoted to replication, which evolves as peer servers come and go.

DS directory data, however, must remain mutable to support write operations. As long as you separate directory data from the configuration, this does not prevent you from replacing directory server replicas. As described in "Initializing Replicas" in the *Administration Guide*, new replicas can start with existing data sets.

Fine-Grained Data Access

DS servers provide both HTTP and LDAP access to directory data. As detailed in "*Request Handling*" in the *Reference*, HTTP access to directory data eventually translates to LDAP access internally. At the LDAP level, DS servers provide powerful, fine-grained access control.

The default server behavior is to refuse all access. All DS servers therefore grant some level of access through privileges as described above, and through access controls. For details, see "*Securing Access to Directory Data*" in the *Security Guide*.

Access control instructions (ACIs) in diretory data take the form of `aci` LDAP attributes, or `global-aci` properties in the server configuration. You write ACIs in a domain-specific language, described in "About ACIs" in the *Administration Guide*. The language lets you describe concisely who has access to what under what conditions. When configuring access control, notice that access controls apply beneath their location in the directory information tree. As a result, some ACIs, such as those granting access to LDAP controls and extended operations, must be configured for the entire server rather than a particular location in the data.

Limiting Privileges

Administrative privileges provide a mechanism that is separate from access control to restrict what administrators can do.

You assign privileges to users as values of the `ds-privilege-name` LDAP attribute. You can assign privileges to multiple users with collective attribute subentries. For more information, see "About Privileges" in the *Administration Guide*.

Take care when granting privileges, especially the following privileges:

- `bypass-acl`: The holder is not subject to access control.
- `config-write`: The holder can edit the server configuration.
- `modify-acl`: The holder can edit access control instructions.
- `privilege-change`: The holder can edit administrative privileges.
- `proxied-auth`: The holder can make requests on behalf of another user, including directory superusers such as `cn=Directory Manager`.

Authentication Methods

DS servers support a variety of authentication mechanisms.

When planning your service, use the following guidelines:

- Limit anonymous access to public data.
- Allow simple (username and password) authentication only over secure connections.
- Require client applications to authenticate based on public key certificates (EXTERNAL SASL mechanism) rather than simple authentication where possible.

For additional suggesions and details, see "*Securing Authentication*" in the *Security Guide*.

Proxy Layer

DS directory proxy servers, and DS DSML and REST to LDAP gateway applications encapsulate DS directory services, and offer access to other directory services.

Unlike directory servers, directory proxy servers do not hold directory data, and so use global access policies rather than ACIs. You define global access policies as server configuration objects. For details, see "About Global Access Control Policies" in the *Administration Guide*.

As mentioned in "Directory Proxy System Resources", be aware that for high-performance services you may need to deploy as many proxy servers or gateways as directory servers.

For details about DS LDAP proxy services, see "*Configuring LDAP Proxy Services*" in the *Administration Guide*.

HTTP Access

As described in "RESTful Client Access Over HTTP" in the *Administration Guide*, you can configure DS servers to provide RESTful HTTP access.

If you deploy this capability, you must configure how HTTP resources map to LDAP entries. This is explained in "Preparing Data Views". For details, see "*REST to LDAP Configuration*" in the *Reference*.

Higher-Level Abstraction

Although LDAP and RESTful HTTP access ensure high performance, your deployment may require a higher level of abstraction than LDAP or HTTP can provide.

Other ForgeRock Identity Platform™ components offer such higher-level abstractions. For example, ForgeRock Access Management software lets you plug into directory services for authentication and account profiles, and then orchestrate powerful authentication and authorization scenarios. ForgeRock Identity Management software can plug into directory services to store configuration and account profiles, to provide user self-services, and to synchronize data with a wide variety of third-party systems.

For an introduction to the alternatives, read the *Identity Platform Guide*.

Replicating Data

Replication is the process of synchronizing data updates across directory servers. Replication is the feature that makes the directory a highly available distributed database.

Trading Consistency for Availability

Replication is designed to provide high availability with tolerance for network partitions. In other words, the service continues to allow both read and write operations when the network is down. Replication provides eventual consistency, not immediate consistency.

According to what is called the CAP theorem, it appears to be impossible to guarantee consistency, availability, and partition tolerance when network problems occur. The CAP theorem makes the claim that distributed databases can guarantee at most two of the following three properties:

Consistency

Read operations reflect the latest write operation (or result in errors).

Availability

Every correct operation receives a non-error response.

Partition Tolerance

The service continues to respond even when the network between individual servers is down or operating in degraded mode.

When the network connection is down between two replicas, replication is temporarily interrupted. Client applications continue to receive responses to their requests[1], but clients making requests

[1] Simultaneous write operations can result in conflicts. Replication resolves most conflicts automatically, but some naming conflicts cannot be resolved automatically. For details, see "Resolving Replication Conflicts" in the *Administration Guide*.

to different servers will not have the same view of the latest updates. The discrepancy in data on different replicas also arises temporarily when a high update load takes time to fully process across all servers.

Eventual consistency can be a trap for the unwary. The client developer who tests software only with a single directory server might not notice problems that become apparent when a load balancer spreads requests evenly across multiple servers. A single server is immediately consistent for its own data. Implicit assumptions about consistency therefore go untested.

For example, a client application that implicitly assumes immediate consistency might perform a write quickly followed by a read of the same data. Tests are all successful when only one server is involved. In deployment, however, a load balancer distributes requests across multiple servers. When the load balancer sends the read to a replica that has not yet processed the write, the client application appears to perform a successful write, followed by a successful read that is inconsistent with the write that succeeded!

When deploying replicated DS servers, keep this eventual consistency trap in mind. Educate developers about the trade off, review patches, and test and fix client applications under your control. In deployments with client applications that cannot be fixed, use affinity load balancing in DS directory proxy servers to work around broken clients. For details, see "Choosing Load Balancing Settings" in the *Administration Guide*.

Deploying Replication

In DS software, the role of a replication server is to transmit messages about updates. Directory servers receive replication messages from replication servers, and apply updates accordingly, meanwhile serving client applications.

Deploy at least two replication servers per local network in case one fails, and deploy more if you have many directory servers per replication server. For LAN-based deployments, each directory server can double as a replication server. For large, WAN-based deployments, consider using standalone replication servers and directory servers.

In a widely distributed deployment, be aware that replication servers all communicate with each other. Directory servers always communicate through replication servers, even if the replication service runs in the same server process as the directory server. By assigning servers to replication groups, you can ensure that directory servers only connect to local replication servers until they need to fail over to remote replication servers. This limits the WAN replication traffic to messages between replication servers, except when all replication servers on the LAN are down. For details, see "Standalone Replication Servers" in the *Administration Guide* and "Replication Groups" in the *Administration Guide*.

Deploy the replication servers first. You can think of them as providing a network service (replication) in the same way DNS provides a network service (name resolution). You therefore install and start replication servers before you add directory servers.

After you install a directory server and configure it as a replica, you must initialize it to the current replication state. There are a number of choices for this, as described in "Initializing Replicas" in the

Administration Guide. Once a replica has been initialized, replication eventually brings its data into a consistent state with the other replicas. As described in "Trading Consistency for Availability", give a heavy update load or significant network latency, temporary inconsistency is expected. You can monitor the replication status to estimate when replicas will converge on the same data set.

Client applications can adopt best practices that work with eventual consistency as described in "*Best Practices For Application Developers*" in the *Developer's Guide*, "Modify: Using Optimistic Concurrency" in the *Developer's Guide*, and "Updating Resources" in the *Developer's Guide*. To work around broken client applications that assume immediate consistency, use affinity load balancing in DS directory proxy servers. For details, see "Choosing Load Balancing Settings" in the *Administration Guide*.

Some client applications need notifications when directory data changes. Client applications cannot participate in replication itself, but can get change notifications. For details, see "Change Notification For Your Applications" in the *Administration Guide*.

Scaling Replication

When scaling replicated directory services, keep the following rules in mind:

- Read operations affect only one replica.

  To add more read performance, use more powerful servers or add servers.

- Write operations affect all replicas.

  To add more write performance, use more powerful servers or add separate replication domains.

When a replica writes an update to its directory data set, it transmits the change information to its replication server for replay elsewhere. The replication server transmits the information to connected directory servers, and to other replication servers replicating the same data. Those replication servers transmit the message to others until all directory servers have received the change information. Each directory server must process the change, reconciling it with other change information.

As a result, you cannot scale up write capacity by adding servers. Each server must replay all the writes.

If necessary, you can scale up write capacity by increasing the capacity of each server (faster disks, more powerful servers) or by splitting the data into separate sets that you replicate independently (data distribution).

Using Replication For Higher Availability

In shared directory service deployments, the directory must continue serving client requests during maintenance operations, including service upgrades, during network outage recovery, and in spite of system failures.

DS replication lets you build a directory service that is always online. DS directory proxy capabilities enable you to hide maintenance operations from client applications. You must still plan appropriate use of these features, however.

As described above, replication lets you use redundant servers and systems that tolerate network partitions. Directory server replicas continue to serve requests when peer servers fail or become unavailable. Directory proxy servers route around directory servers that are down for maintenance or down due to failure. When you upgrade the service, you roll out one upgraded DS server at a time. New servers continue to interoperate with older servers, so the whole service never goes down. All of this depends on deploying redundant systems, including network links, to eliminate single points of failure. For more, see "Building a Highly Available Directory".

As shown in that section, your deployment may involve multiple locations, with servers communicating locally over LANs and remotely over WANs. Your deployment can also use separate replication topologies, for example, in order to sustain very high write loads, or to separate volatile data from more static data. Carefully plan your load balancing strategy to offer good service at a reasonable cost. By using replication groups, you can limit most replication traffic over WAN links to communications between replication servers. Directory proxy servers can direct client traffic to local servers until it becomes necessary to failover to remote servers.

Sound operational procedures play as important a role in availability as good design. Operational staff maintaining the directory service must be well-trained and organized so that someone is always available to respond if necessary. They must have appropriate tools to monitor the service in order to detect situations that need attention. When maintenance, debugging, or recovery is required, they should have a planned response in most cases. Your deployment plans should therefore cover the requirements and risks that affect your service, as described in the other sections of this chapter.

Before finalizing deployment plans, make sure that you understand key availability features in detail. For details about replication, read "*Managing Data Replication*" in the *Administration Guide*. For details about proxy features, read "*Configuring LDAP Proxy Services*" in the *Administration Guide*.

Preparing for Backup and Recovery

Make sure your plans define how you:

- Back up directory data
- Safely store backup archives
- Recover your directory service from backup archives

DS servers store data in *backends*. A backend is a private server repository that can be implemented in memory, as a file, or as an embedded database. DS servers use local backends to store directory data, server configuration, LDAP schema, and administrative tasks. Directory proxy servers implement a type of backend for non-local data, called a proxy backend, which forwards LDAP requests to a remote directory service.

For performance reasons, DS servers store directory data in a local database backend, which is a backend implemented using an embedded database. Database backends are optimized to store directory data. Database backends hold data sets as key-value pairs. LDAP objects fit the key-value model very effectively, with the result that a single database backend can serve hundreds of millions of LDAP entries. Database backends support indexing and caching for fast lookups in large data sets. Database backends do not support relational queries or direct access by other applications. For more information, see "About Database Backends" in the *Administration Guide*.

Backup and restore procedures are described in "About Database Backends" in the *Administration Guide*. When planning your backup and recovery strategies, be aware of the following key features:

- Backup and restore tasks can run while the server is online. They can, however, have a significant impact on server performance.

  For deployments with high performance requirements, consider dedicating a replica to perform only backup operations. This prevents other replicas from stealing cycles to back up data that could otherwise be used to serve client applications.

- When you restore data from a recent backup archive on a directory server replica, the replication protocol brings the replica up to date with others after the restore operation.

  The requires, however, that the backup is recent enough. Backup archives older than the replication purge delay (default: 3 days) are stale and should be discarded.

- Directory data replication ensures that all servers converge on the latest data. If your data is affected by an serious accidental deletion or change, you must restore the entire directory service to an earlier state.

  For details, see "Recovering From User Error" in the *Administration Guide*.

- When you restore encrypted backup data on a directory server, the server must first be properly configured as a replica of the server that performed the backup. This is also the case if passwords are stored with a reversible encryption password storage scheme.

  Otherwise the directory server cannot obtain the symmetric key for decryption. For details, see "Encrypting Directory Data" in the *Administration Guide*.

- In a disaster situation and as an alternative to binary backup archives, you can export directory data to LDIF files. You can then restore directory data by importing the LDIF.

  Be aware if you have stored passwords with a reversible encryption password storage scheme that the server must first be properly configured as a replica of the server where you exported the data. Otherwise the directory server cannot obtain the symmetric key for decryption.

- You can perform a file system backup of your servers instead of using the server tools.

  You must, however, *stop the server before taking a file system backup*. Running DS directory servers cannot guarantee that database backends will be recoverable unless you back them up with the DS tools.

Monitoring and Auditing the Directory Service

When monitoring DS servers and auditing access, be aware that you can obtain some but not all data remotely.

The following data sources allow remote monitoring:

- HTTP connection handlers expose a `/metrics/api` endpoint that offers RESTful access to monitoring data, and a `/metrics/prometheus` endpoint for Prometheus monitoring software.

  For details, see "To Set Up REST Access to Administrative Data" in the *Administration Guide*.
- LDAP connection handlers expose a `cn=monitor` branch that offers LDAP access to monitoring data.

  For details, see "LDAP-Based Monitoring" in the *Administration Guide*.
- JMX connection handlers offer remote access.

  For details, see "JMX-Based Monitoring" in the *Administration Guide*.
- SNMP connection handlers enable remote access.

  For details, see "SNMP-Based Monitoring" in the *Administration Guide*.
- You can configure alerts to be sent over JMX or SMTP (mail).

  For details, see "Alert Notifications" in the *Administration Guide*.
- Replication conflicts are found in the directory data.

  For details, see "Resolving Replication Conflicts" in the *Administration Guide*.
- Server tools, such as the **status** command, can run remotely.

  For details, see "Server Operation and Tasks" in the *Administration Guide*.

The following data sources require access to the server system:

- Server logs, as described in "Server Logs" in the *Administration Guide*.

  DS servers write log files to local disk subsystems. In your deployment, plan to move access logs that you want to retain. Otherwise the server will eventually remove logs according to its retention policy to avoid filling up the disk.
- Index verification output and statistics, as described in "Rebuilding Indexes" in the *Administration Guide*, and "Verifying Indexes" in the *Administration Guide*.

When defining how to monitor the service, use the following guidelines:

- Your service level objectives (SLOs) should reflect what your stakeholders expect from the directory service for their key client applications.

If SLOs reflect what stakeholders expect, and you monitor them in the way key client applications would experience them, your monitoring system can alert operational staff when thresholds are crossed, before the service fails to meet SLOs.

- Make sure you keep track of resources that can expire, such as public key certificates and backup archives from directory server replicas, and resources that can run out, such as system memory and disk space.
- Monitor system and network resources in addition to the directory service.

  Make sure that operational staff can find and fix problems with the system or network, not only the directory.
- Monitor replication delay so you can take action when it rises above a reasonable threshold.

In order to analyze server logs, use other software, such as Splunk, which indexes machine-generated logs for analysis.

If you require integration with an audit tool, plan the tasks of setting up logging to work with the tool, and analyzing and monitoring the data once it has been indexed. Consider how you must retain and rotate log data once it has been consumed, as a high-volume service can produce large volumes of log data.

Hardening and Securing the Directory Service

When you first set up DS servers for evaluation, the configuration favors ease of use over security.

When preparing for deployment, consider using the `setup --productionMode` option. This option hardens the server for more security. When you use it, however, you must explicitly grant access that is available by default in evaluation mode.

Setting up a server in hardened production mode leads to the following settings:

- The default backend database for directory servers, `userRoot`, uses data confidentiality to encrypt potentially sensitive data on disk.
- Global access control allows only the following access:
  - Anonymous users can request the StartTLS extended operation, and the Get Symmetric Key extended operation. The Get Symmetric Key extended operation is an operation designed for DS for internal use. DS servers require Get Symmetric Key extended operation access to create and share secret keys for encryption.
  - Anonymous users can read the root DSE operational attributes that describe server capabilities, including among other information, what security protocols and cipher suites the server supports.
  - Authenticated users can read the LDAP directory schema.

  - Authenticated users can request the LDAP Password Modify extended operation, the Who am I? extended operation, and the Cancel extended operation.
  - Authenticated users can request the Pre-Read and Post-Read controls, the Subtree Delete control, and the Permissive Modify control. These controls are used by the REST to LDAP gateway.

    Authenticated users can also request the ForgeRock Transaction ID control. This is a ForgeRock-specific control for internal use that permits transmission of transaction IDs through platform components for use as a key to correlation of Common Audit events.
  - If the setup process creates a monitor user, this user is granted access to read monitoring data.

  For a longer explanation of these settings, see "Reconsider Default Global Access Control" in the *Security Guide*.

- The protocol version and cipher suites for securing connections are restricted to those using strong encryption.

  The protocol version is restricted to `TLSv1.2`.

  The cipher suites used when negotiating a secure connection call for a server certificate using an elliptic curve (EC) key algorithm or an RSA key algorithm. If you provide your own keystore when setting up the server in production mode, make sure that the certificate key algorithm is EC or RSA. Otherwise the server will not be able to negotiate secure connections. For details and examples, see "To Restrict Protocols and Cipher Suites" in the *Security Guide*.

- The Crypto Manager requires encrypted communication between servers.

  The Crypto Manager is described in "Cryptographic Key Management" in the *Security Guide*.

- The anonymous HTTP authorization mechanism for REST access is disabled.

  As a result, REST access does not permit anonymous requests.

- DS native file-based access loggers and the replication error logger have UNIX/Linux file permissions set to `600` (only the server account has read-write access to log files). This setting does not affect Common Audit loggers, such as the JSON file-based audit loggers.

  Adjust system settings to ensure appropriate access to files. For additional information and recommendations on setting the UNIX/Linux `umask` appropriately and on setting ACLs on Windows systems, see "Setting Appropriate File Permissions" in the *Security Guide*.

- The random password generator generates 10-character alphanumeric passwords.

- The default password policy for normal users requires passwords at least 8 characters in length, and prevents use of common passwords.

  The password policy for the default directory superuser requires passwords at least 8 characters in length, prevents use of common passwords, and requires that authentication be secure to avoid exposing credentials over the network.

- The CRAM-MD5 and DIGEST-MD5 SASL mechanisms are disabled.

For additional details, see "*Securing and Hardening Servers*" in the *Administration Guide*.

Planning Tests

In addition to planning tests for each custom plugin, test each feature you deploy. Perform functional and also non-functional testing to validate that the directory service meets SLOs under load in realistic conditions. Include acceptance tests for the actual deployment. The data from the acceptance tests help you to make an informed decision about whether to go ahead with the deployment or to roll back.

Testing Functional Capabilities

Functional testing validates that specified test cases work with the software considered as a black box.

As ForgeRock already tests DS servers and gateways functionally, focus your functional testing on customizations and service level functions. For each key capability, devise automated functional tests. Automated tests make it easier to integrate new deliveries to take advantage of recent bug fixes and to check that fixes and new features do not cause regressions.

As part of the overall plan, include not only tasks to develop and maintain your functional tests, but also to provision and to maintain a test environment in which you run the functional tests before you significantly change anything in your deployment. For example, run functional tests whenever you upgrade any server or custom component, and analyze the output to understand the effect on your deployment.

Testing Service Performance Testing

With written SLOs, even if your first version consists of guesses, you turn performance plans from an open-ended project to a clear set of measurable goals for a manageable project with a definite outcome. Therefore, start your testing with service level objectives clear definitions of success.

Also, start your testing with a system for load generation that can reproduce the traffic you expect in production, and underlying systems that behave as you expect in production. To run your tests, you must therefore generate representative load data and test clients based on what you expect in production. You can then use the load generation system to perform iterative performance testing.

Iterative performance testing consists of identifying underperformance and the bottlenecks that cause it, and discovering ways to eliminate or work around those bottlenecks. Underperformance means that the system under load does not meet service level objectives. Sometimes resizing or tuning the system can help remove bottlenecks that cause underperformance.

Based on SLOs and availability requirements, define acceptance criteria for performance testing, and iterate until you have eliminated underperformance.

Tools for running performance testing include the tools listed in "Testing Performance" in the *Administration Guide*, and Gatling, which uses a domain specific language for load testing. To mimic the production load, examine both the access patterns and also the data that DS servers store. The representative load should reflect the distribution of client access expected in production.

Although you cannot use actual production data for testing, you can generate similar test data using tools, such as the **makeldif** command.

As part of the overall plan, include not only tasks to develop and maintain performance tests, but also to provision and to maintain a pre-production test environment that mimics your production environment. Security measures in your test environment must also mimic your production environment, as security measures can impact performance.

Once you are satisfied that the baseline performance is acceptable, run performance tests again when something in your deployment changes significantly with respect to performance. For example, if the load or number of clients changes significantly, it could cause the system to underperform. Also, consider the thresholds that you can monitor in the production system to estimate when your system might start to underperform.

Testing the Deployment

Deployment testing is a description rather than a term in the context of this guide. It refers to the testing implemented within the deployment window after the system is deployed to the production environment, but before client applications and users access the system.

Plan for minimal changes between the pre-production test environment and the actual production environment. Then test that those changes have not cause any issues, and that the system generally behaves as expected.

Take the time to agree upfront with stakeholders regarding the acceptance criteria for deployment tests. When the production deployment window is small, and you have only a short time to deploy and test the deployment, you must trade off thorough testing for adequate testing. Make sure to plan enough time in the deployment window for performing the necessary tests and checks.

Include preparation for this exercise in your overall plan, as well as time to check the plans close to the deployment date.

Managing Configuration Changes

Make sure your plan defines the change control process for configuration. Identify the ways that the change is likely to affect your service. Validate your expectations with appropriate functional, integration, and stress testing. The goal is to adapt how you maintain the service before, during, and after the change. Complete your testing before you subject all production users to the change.

The rest of this section covers DS server and gateway configuration options, so you know what must be under change control.

Server Configuration

DS servers store configuration in files under the server's `config` directory. When you set up a server, the setup process creates the initial configuration files based on templates in the server's `template` directory. "*File Layout*" in the *Reference* describes the files.

When a server starts, it reads its configuration files to build an object view of the configuration in memory. This view holds the configuration objects, and the constraints and relationships between objects. This view of the configuration is accessible over client-side and server-side APIs. Configuration files provide a persistent, static representation of the configuration objects.

Configuration tools use the client-side API to discover the server configuration and to check for constraint violations and missing relationships. The tools prevent you from breaking the server configuration structurally by validating structural changes before applying them. The server-side API allows the server to validate configuration changes, and to synchronize the view of the configuration in memory with the file representation on disk. If you make changes to the configuration files on disk while the server is running, the server can neither validate the changes beforehand, nor guarantee that they are in sync with the view of the configuration in memory.

"DS Server Configuration" summarizes the configuration options.

DS Server Configuration

Method	**Notes**
Tools (**dsconfig** and others)	Stable, supported, public interfaces for editing server configurations. Most tools work with local and remote servers, both online and offline.
Files	Internal interface to the server configuration, subject to change without warning in any release. If you must make manual changes to configuration files, always stop the DS server before editing the files. If the changes break the configuration, compare with the `var/config.ldif.startok` file, and with the compressed snapshots of the main configuration in the `var/archived-configs/` directory.
REST (HTTP) API	Internal interface to the server configuration, subject to change without warning in any release. Useful for enabling browser-based access to the configuration.

Once a server begins to replicate data with other servers, the part of the configuration pertaining to replication is specific to that server. As a result, a server effectively cannot be cloned once it has begun to participate in data replication. When deploying servers, do not initialize replication until you have deployed the server.

Gateway Configuration

You edit files to configure DS DSML and REST to LDAP gateway web applications.

The gateways do not have external configuration APIs, and must be restarted after you edit configuration files for the changes to take effect.

Developing a Documentation Plan

The DS product documentation is written for readers like you, who are architects and solution developers, as well as for DS developers and for administrators who have had DS training. The people operating your production environment need concrete documentation specific to your deployed solution, with an emphasis on operational policies and procedures.

Procedural documentation can take the form of a runbook with procedures that emphasize maintenance operations, such as backup, restore, monitoring and log maintenance, collecting data pertaining to an issue in production, replacing a broken server or web application, responding to a monitoring alert, and so forth. Make sure in particular that you document procedures for taking remedial action in the event of a production issue.

Furthermore, to ensure that everyone understands your deployment and to speed problem resolution in the event of an issue, changes in production must be documented and tracked as a matter of course. When you make changes, always prepare to roll back to the previous state if the change does not perform as expected.

Developing a Maintenance and Support Plan

If you own the architecture and planning, but others own the service in production, or even in the labs, then you must plan coordination with those who own the service.

Start by considering the service owners' acceptance criteria. If they have defined support readiness acceptance criteria, you can start with their acceptance criteria. You can also ask yourself the following questions:

- What do they require in terms of training in DS software?
- What additional training do they require to support your solution?
- Do your plans for documentation and change control, as described in "Developing a Documentation Plan", match their requirements?
- Do they have any additional acceptance criteria for deployment tests, as described in "Testing the Deployment"?

Also, plan back line support with ForgeRock or a qualified partner. The aim is to define clearly who handles production issues, and how production issues are escalated to a product specialist if necessary.

Include a task in the overall plan to define the hand off to production, making sure there is clarity on who handles monitoring and issues.

Creating a Rollout Plan

In addition to planning for the hand off of the production system, also prepare plans to roll out the system into production. Rollout into production calls for a well-choreographed operation, so these are likely the most detailed plans.

Take at least the following items into account when planning the rollout:

- Availability of all infrastructure that DS software depends on, such as the following:
  - Server hosts and operating systems
  - Web application containers for gateways
  - Network links and configurations
  - Persistent data storage
  - Monitoring and audit systems
- Installation for all DS servers.
- Final tests and checks.
- Availability of the personnel involved in the rollout.

In your overall plan, leave time and resources to finalize rollout plans toward the end of the project.

Planning for Change

To succeed, your directory service must adapt to changes, some that you can predict, some that you cannot.

In addition to the configuration changes covered in "Managing Configuration Changes", predictable changes include the following:

Increases and decreases in use of the service

For many deployments, you can predict changes in the use of the directory service, and in the volume of directory data.

If you expect cyclical changes, such as regular batch jobs for maintenance or high traffic at particular times of the year, test and prepare for normal and peak use of the service. For deployments where the peaks are infrequent but much higher than normal, it may be cost effective to dedicate replicas for peak use that are retired in normal periods.

If you expect use to increase permanently, then decide how much headroom you must build into the deployment. Plan to monitor progress and add capacity as necessary to maintain headroom, and to avoid placing DS servers under so much stress that they stop performing as expected.

If you expect use to decrease permanently, at some point you will retire the directory service. Make sure all stakeholders have realistic migration plans, and that their schedules match your schedule for retirement.

Depending on the volume of directory data and the growth you expect for the directory service, you may need to plan for scalability beyond your initial requirements.

As described in "Scaling Replication", you can increase read performance by adding servers. To increase write performance, adding servers is not the solution. Instead, you must split the data into sets that you replicate separately.

Luckily, single directory services can already support thousands of replicated write operations per second, meaning millions of write operations per hour. It may well be possible to achieve appropriate performance by deploying on more powerful servers, and by using higher performance components, such as dedicated SSD disks instead of traditional disks.

When scaling up the systems is not enough, you must instead organize your DIT so that you can replicate different branches separately. Deploy the replicas for each branch on sets of separate systems. For details, see "Building a High-Scale Directory".

Directory service upgrades

ForgeRock regularly offers new releases of DS software. These include maintenance and feature releases. Supported customers may also receive patch releases for particular issues.

Patch and maintenance releases are generally fully compatible. Plan to test and roll out patch and maintenance releases swiftly, as they include important updates such as fixes for security issues or bugs that you must address quickly.

Plan to evaluate feature releases as they occur. Even if you do not intend to use new features immediately, you might find important improvements that you should roll out. Furthermore, by upgrading regularly you apply fewer changes at a time than you would by waiting until the end of support life and then performing a major upgrade.

Key rotation

Even if you do not change the server configuration, the signatures eventually expire on certificates used to secure connections. You must at minimum replace the certificates and restart the connection handlers that use them. You could also change the key pair in addition to getting a new certificate.

If you encrypt directory data for confidentiality, you might also choose to rotate the symmetric encryption key.

Unpredictable changes include the following:

Disaster recovery

As described in "Using Replication For Higher Availability", assess the risks. In light of the risks, devise and test disaster recovery procedures.

New security issues

Time and time again, security engineers have found vulnerabilities in security mechanisms that could be exploited by attackers. Expect this to happen during the lifetime of your deployment.

You might need to change the following at any time:

- Keys used to secure connections
- Keys used to encrypt directory data
- Protocol versions used to secure connections
- Password storage schemes
- Deployed software that has a newly discovered security bug

In summary, plan to adapt your service to changing conditions. To correct security bugs and other issues and to recover from minor or major disasters, be prepared to patch, upgrade, roll out, and roll back changes as part of your regular operations.

Chapter 4
Applying Deployment Patterns

This chapter proposes appropriate patterns for specific deployments.

Embedding the Directory in a Java Application

As described in "*Embedding the Server*" in the *Developer's Guide*, your Java application can include an embedded directory service. The application manages the embedded service through the server process API.

When you embed a server in your Java application, consider the following deployment choices:

- **Is an embedded server the best choice?**

  On one hand, an embedded server runs in the same memory space as your application. It uses file system and processor resources that your application could use. By default, an embedded directory server uses up to 2 GB of disk space for logs, beyond the disk space used for your directory data. By default, each database backend requests 50% of JVM memory. The default settings, such as memory use, are adjustable. But the size of directory data on disk depends on your data.

  On the other hand, your application has full control over the service. The embedded server can be available whenever your application is up. It presents a consistent view of its data to your application. In addition, DS servers make replicating data to another server straightforward if you find this is necessary.

- **How do you change server settings?**

  With the configuration API, you can make changes while the server is online. The configuration API can validate changes, and return errors for changes that would break the configuration.

  If you manage server settings through changes to files, you must take the server offline to make configuration changes. Taking the server offline for every configuration change does ensure, however, that the server configuration never changes while it is running.

- **How do you manage log and backup files?**

  By default, the server writes an access log message to a file for each client request. Also, by default, the server eventually rotates and retains log files, but the default settings are designed for standalone server deployments. For details, see "Server Logs" in the *Administration Guide*.

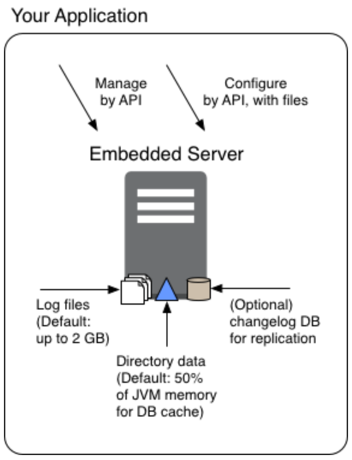
The server commands let you back up and restore directory data to compressed files on local disk. An alternative is to stop the server and perform backup at the file system level. If you back up directory data, you must decide how to avoid filling available space, and how to restore when necessary. If you replicate directory data, account for the fact that backup files become stale after the replication purge delay. For details, see "*Backing Up and Restoring Data*" in the *Administration Guide*.

- **How do you upgrade an embedded server?**

  The upgrade process can require space for extra data. Some upgrades require exporting data to LDIF and importing from LDIF after upgrade.

  You can, however, script the upgrade process to run unattended as part of a general upgrade of your application.

Considerations for Embedded DS Servers


Your Application
Manage by API
Configure by API, with files
Embedded Server
Log files (Default: up to 2 GB)
(Optional) changelog DB for replication
Directory data (Default: 50% of JVM memory for DB cache)

Building a Highly Available Directory

When you deploy DS servers into a highly available directory service, you are implementing the primary use case for which DS software is designed:

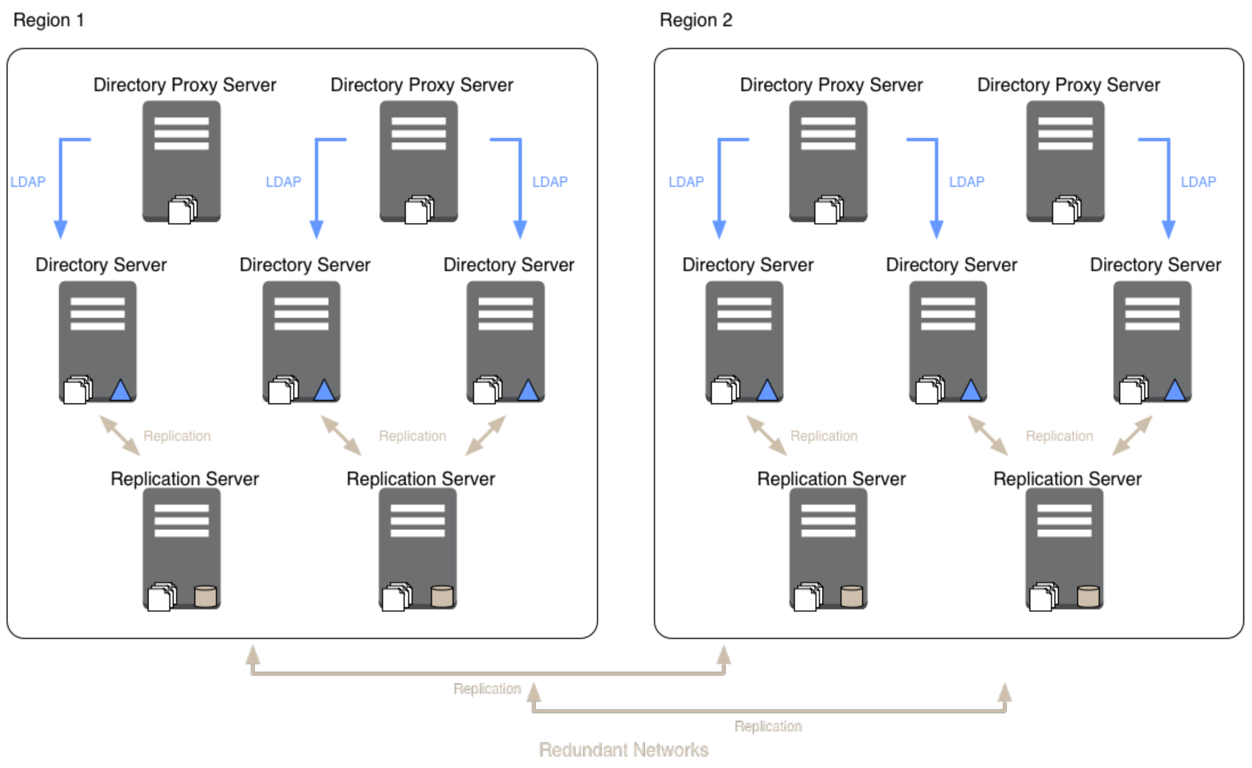
- Data replication lets you eliminate single points of failure.

  When using replication, keep in mind the trade off described in "Trading Consistency for Availability".

- DS upgrade capabilities let you perform rolling upgrades without ever taking the whole service offline.

- DS proxy capabilities help you to provide a single point of entry for directory applications, hiding the fact that individual servers do go offline.

Highly Available Directory Service



You build a highly available directory service by using redundant servers in multiple locations. If possible, use redundant networks within and between locations to limit network partitions. When you install or upgrade a highly available directory service, bring component servers online in the following order:

1. **Replication Servers**

   Replication servers provide the foundation of a highly available DS service. They communicate change messages to directory server replicas, and they also allow other servers, such as DS directory proxy servers, to discover available replicas.

2. **Directory Servers**

Directory server replicas ultimately respond to client application requests. They hold an eventually convergent copy of the directory data. They require a replication service to communicate with other replicas about changes to their copy of the directory data.

3. **Directory Proxy Servers**

   Directory proxy servers provide a unified view of the service to client applications. They discover DS replicas by querying the replication service. They forward requests to the replicas, and responses to the client applications.

In addition to redundant server components, avoiding downtime depends on being able operationally to recover quickly and effectively. Prepare and test your plans. Even if disaster strikes, you will be able to repair the service promptly.

Plan how you store backup files both onsite and offsite. If you are using data confidentiality to encrypt directory data in database backends, take care to retain not only the backup data but also the replicated configuration that enables directory servers to decrypt encrypted directory data. For details, see "*Backing Up and Restoring Data*" in the *Administration Guide*.

When defining disaster recovery plans, consider at least the following situations:

- **The entire service is down.**

  It is important to distinguish whether the situation is temporary and easily recoverable, or permanent and requires implementation of disaster recovery plans.

  If an accident, such as a sudden power cut at a single-site deployment, brought all the servers down temporarily, restart them when the power returns. As described in "Server Recovery" in the *Administration Guide*, directory servers might have to replay their transaction logs before they are ready. But this operation happens automatically when you restart the server.

  In a disaster, the entire service could go offline permanently. Be prepared to rebuild the entire service, initializing replicas by restoring from backup files, or importing from backup LDIF. For details, see "Initializing Replicas" in the *Administration Guide*.

- **Part of the service is down.**

  Failover client applications to servers still in operation, and restart or rebuild servers that are down.

  You can configure directory proxy servers to failover automatically, and to retry requests for certain types of failure. For details, see "*Configuring LDAP Proxy Services*" in the *Administration Guide*.

- **The network is temporarily down between servers.**

  By default, you do not need to take immediate action for a temporary network outage. As long as client applications can still communicate with local servers, replication is designed to catch up when the network connections are reestablished.

By default, when a directory server replica cannot communicate with a replication server, the replication domain `isolation-policy` prevents the directory server replica from accepting updates.

In any case, if the network is partitioned longer than the replication purge delay (default: 3 days), then replication will have purged older data, and might not be able to catch up. For longer network outages, you will have to reinitialize replication.

When defining procedures to rebuild a service that is permanently offline, the order of operations is the same as during an upgrade:

1. Redirect client applications to a location where the service is still running.

   If the proxy layer is still running, directory proxy servers can automatically fail requests over to remote servers that are still running.

2. Rebuild replication servers.

3. Rebuild directory servers.

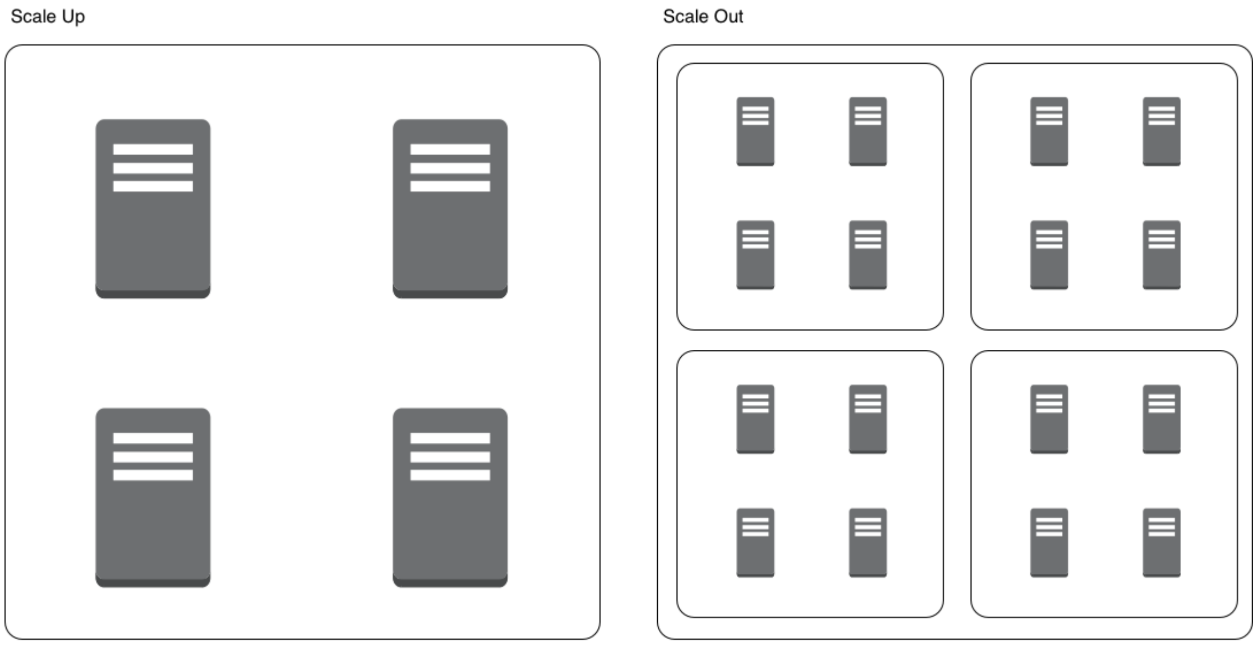
4. Rebuild directory proxy servers.

Building a High-Scale Directory

A high-scale directory service is one that requires high performance. For example, very high throughput or very low response times, or both. Or, it has a large data set, such as 100 million entries. When building a high-scale directory, the fundamental question is whether to scale up or scale out.

Scaling up means deploying more powerful server systems. *Scaling out* means deploying many more server systems.

High-Scale Alternatives



The following table summarizes the reasons to choose one approach over another, and the trade offs to consider for each choice.

Scaling Up vs. Scaling Out

	Scaling Up	**Scaling Out**
Why Choose...?	Simpler architecture Cannot distribute or shard data	Very high update load Can distribute or shard data
Advantages	Simpler architecture No need to distribute or shard data	Not limited by underlying platform Smaller server systems Better isolation of issues High update scalability
Disadvantages	Limited by underlying platform Powerful (expensive) server systems Less isolation of issues	Complex architecture Must distribute/shard data somehow

	Scaling Up	Scaling Out
	Limited write scalability	

Planning for High Scale

Before building a test directory service, start sizing systems by considering service level objectives (SLOs) and directory data.

Define SLOs as described in "Defining Performance Requirements and Constraints" in the *Administration Guide*. Once you have defined the SLOs, model directory client traffic to test them using your own tools or the tools described in "Testing Performance" in the *Administration Guide*.

Estimate the disk space needed for each server. This depends on the traffic you modelled to meet SLOs, and on directory data that represents what you expect in production:

1. Import a known fraction of the expected initial data with the server configured for production.

   For help, see "Generating Test Data" in the *Administration Guide*. Make sure you adapt the template for *your* data. Do not rely only on the default template for the **makeldif** command.

2. Check the size of the database.

   Divide by the fraction used in the previous step to estimate the total starting database size.

3. Multiply the result to account for replication metadata.

   To estimate the volume of replication metadata, set up replication with multiple servers as expected in production, and run the estimated production load that corresponds to the data you used. Keep the load running until the replication purge delay. After the purge delay, measure the size of the databases on a directory server, and the size of the changelog database on a replication server. Assuming the load is representative of the production load including expected peaks and normal traffic, additional space used since the LDIF import should reflect expected growth due to replication metadata.

4. Multiply the result to account for the overall growth that you expect for the directory service during the lifetime of the current architecture.

5. To complete the estimate, add 2 GB for default access log files, and space for any backups or LDIF exports you expect to store on local disk.

For a directory server, estimate the memory required to cache the database, as described in "Database Cache Settings" in the *Administration Guide*.

Scaling Up

When scaling up on appropriately powerful server systems, each system must have the resources to run a high-scale DS server. As described in "Scaling Replication", a directory server replica is only required to absorb its share of the full read load. But each replica must be able to absorb the full write load for the service.

Make sure that the estimates you arrived at in "Planning for High Scale" remain within the capabilities of each server and system.

In addition to the recommendations in "Choosing Hardware" in the *Release Notes*, and the tips in "Tweaking DS Performance" in the *Administration Guide*, consider the following points to avoid resource contention:

- For best performance, use dedicated servers.
- Run as few additional system services as possible.
- Run standalone replication servers, directory servers, and directory proxy servers on separate systems.
- In addition to using fast disks with good IOPS, put logs, databases, and backup files on separate disk subsystems.
- Keep resource limitations for client applications to acceptable minimums.
- Schedule backups and maintenance for minimum service impact.

Scaling Out

When scaling out onto multiple server systems, you must find a usable way to distribute or shard the data into separate replication domains. In some cases, each replication domain holds a branch of the *DIT* with a similar amount of traffic and an equivalent amount of data. Entries could then be distributed based on location or network or some other attribute. Branches could join at a base DN that brings all the entries together in the same logical view.

Separate at least the directory server replicas in each replication domain, so that they share only minimal and top-level entries. To achieve this, use subtree replication, which is briefly described in "Subtree Replication" in the *Administration Guide*. Each replica can hold minimal and top-level entries in one database backend, but its primary database backend holds only the branch it shares with others in the domain.

If the data to scale out is all under a single DN, consider using a DS proxy server layer to perform the data distribution as described in "Scaling Out Using Data Distribution".

When building a scaled-out architecture, be sure to consider the following questions:

- How will you distribute the data to allow the service to scale naturally, for example, by adding a replication domain?
- How will you manage what are essentially multiple directory services?

  All of your operations, from backup and recovery to routine monitoring, must take the branch data into account, always distinguishing between replication domains.

- How will you automate operations?
- How will you simplify access to the service?

  Consider using DS proxy servers for a single point of entry, as described in "*Configuring LDAP Proxy Services*" in the *Administration Guide*, and in particular "Deploying a Single Point of Directory Access" in the *Administration Guide*.

Scaling Out Using Data Distribution

DS directory proxy servers can distribute data across multiple shards. Proxy settings are described in "*Configuring LDAP Proxy Services*" in the *Administration Guide*. This section shows you how to try data distribution for AM Core Token Service (CTS) tokens.

Data distribution through the proxy comes with the following constraints:

- Data distribution is not elastic, and does not redistribute data if you add a shard. Once you deploy and use data distribution, you cannot change the number of shards.

  Plan for peak load when using data distribution to scale out your deployment.

  You can, however, add or change individual servers within a shard. For example, you could scale out by deploying many shards, and later upgrade to more powerful, faster underlying systems to scale up each shard.

- You cannot import distributed data from LDIF. The **import-ldif** command does not work with a proxy backend.

  If you must initialize distributed data, add the entries through the proxy server instead. You can use the **ldapmodify** command, for example, setting the `--numConnections` option to perform updates in parallel on multiple LDAP connections. You can also deploy as many proxy servers as necessary to perform the adds in parallel.

- Write requests for entries above the distributed data are not repeated on all shards. Instead the proxy rejects such requests.

  If you must make a change to entries above the distributed data, make the change behind the proxy to one directory server replica in each shard.

- Subtree and one-level searches can be relatively expensive, as the proxy must potentially forward the search request to every shard to retrieve all search results.

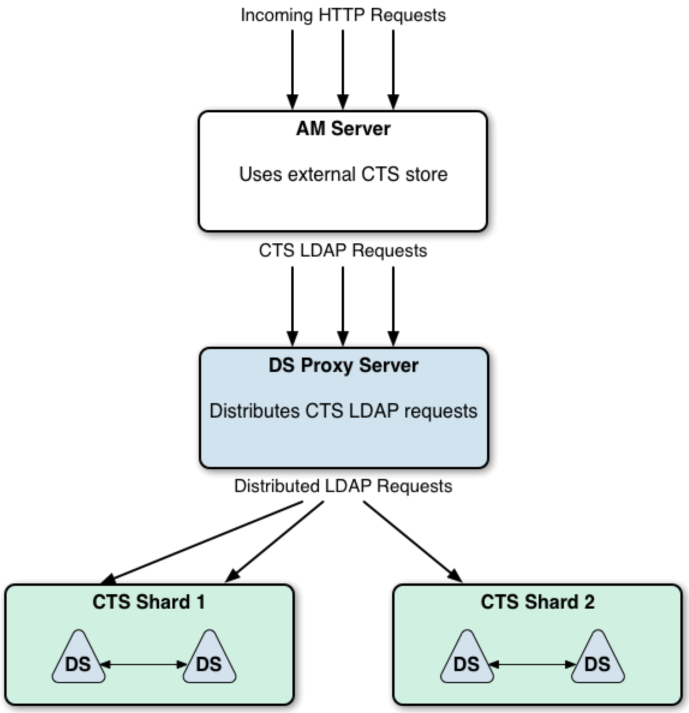
  To optimize searches, use a search base DN that points to a distributed entry.

- Given the present constraints of both data distribution and replication, the best fit for data distribution occurs where the distributed data remains self-contained and logically independent from other data branches.

  The example shown below distributes data for AM CTS tokens. The CTS data model fits DS data distribution well. AM stores all CTS token entries under the same DN. The entries above the distributed CTS tokens do not change during normal operation. CTS token entries do not have DN-value attributes that reference entries in another branch, nor are they members of LDAP groups stored in another branch.

The example that follows is intended for evaluation on a single computer. It involves installing five DS servers and one AM server:

Sample CTS Store With Data Distribution



As you follow the example, notice the following characteristics:

- The example is intended for evaluation only.

  In production, deploy each server on a separate system, set up DS servers in production mode, and use secure connections.

- The AM server connects to the DS proxy for CTS LDAP requests.

- The DS proxy distributes LDAP requests across two shards.

- Each shard holds two DS replicas set up with the AM CTS profile.

DS servers replicate only *within* shards. Servers in different partitions never replicate to each other. If replication crosses a partition boundary, the data is replicated everywhere. Replicating data everywhere would defeat the purpose of data distribution.

To Try the CTS Data Distribution Example

Follow these steps:

1. Make sure you have the Bash shell installed so that you can run the scripts to install and configure DS servers.
2. Download the DS .zip delivery and keep track of where you saved it.

   The scripts use the file name, `~/Downloads/DS-6.5.6.zip`.
3. Stop services that use the following ports. Servers in the example cannot start if these ports are in use:

   `1389`
   `4444`
   `5389`
   `5444`
   `5989`
   `6389`
   `6444`
   `6989`
   `8080`
   `15389`
   `15444`
   `15989`
   `16389`
   `16444`
   `16989`
4. Set up the DS replicas:

   a. Save a local copy of the script, .

      The script is shown in the following listing:

```
#!/usr/bin/env bash
#
# Copyright 2018 ForgeRock AS. All Rights Reserved
#
# Use of this code requires a commercial software license with ForgeRock AS.
# or with one of its affiliates. All use shall be exclusively subject
# to such license between the licensee and ForgeRock AS.
#

###
# Set up directory server replicas for CTS in appropriate shards.
# This is intended for evaluation on a single system.
#
# In deployment, each of these replicas would be on a separate host system.
#
# The proxy distributes data across two CTS shards, each with two replicas.
#
# Each shard is served by a pool of separate replicas.
# In other words, each server replicates within one shard only.
#
# All servers have the same uid=Proxy service account.
# The proxy binds with uid=Proxy and uses proxied authorization.
# This script adds an ACI to allow the proxy service account
# to perform proxied authorization under the CTS shards.
#
# All CTS shards have a CTS admin account,
# uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens.
# To modify the account, for example to change the password,
# you must modify it once for each shard, not through the proxy.
# The proxy would distribute the change to only one shard.
#
# The shards have the following replication configurations,
# with each server's (admin-port ldap-port):
# cts 1: (5444 5389) <==> (15444 15389)
# cts 2: (6444 6389) <==> (16444 16389)
###

###
# Adjust these variables to fit your circumstances:
ZIP=~/Downloads/DS-6.5.6.zip
BASE_DIR=/path/to
FQDN=cts.example.com
TEMP_DIR=/tmp
###

CURRENT_DIR=$(pwd)
INSTANCE_PREFIX=ds-rs-

# Write LDIF for proxy service account.
create_ldif() {
  cat > ${TEMP_DIR}/proxyUser.ldif << EOF
dn: uid=proxy
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Proxy
sn: User
ds-privilege-name: proxied-auth
userPassword: password
EOF
}

# Install a single DS/RS with the CTS profile from the .zip distribution.
# The AM CTS reaper will manage token expiration and deletion.
# $1: instance number
install() {
  echo "### Installing ${BASE_DIR}/${INSTANCE_PREFIX}${1} ###"
  unzip -q ${ZIP}
  mv opendj ${INSTANCE_PREFIX}${1}
  ${BASE_DIR}/${INSTANCE_PREFIX}${1}/setup \
  directory-server \
   --adminConnectorPort ${1}444 \
   --hostname ${FQDN} \
   --ldapPort ${1}389 \
   --enableStartTls \
   --rootUserDN "cn=Directory Manager" \
   --rootUserPassword password \
   --profile am-cts \
   --set am-cts/amCtsAdminPassword:password \
   --acceptLicense

  ${BASE_DIR}/${INSTANCE_PREFIX}${1}/bin/dsconfig \
  set-global-configuration-prop \
   --bindDN "cn=Directory Manager" \
   --bindPassword password \
   --hostname ${FQDN} \
   --port ${1}444 \
   --set server-id:${1} \
   --trustAll \
   --no-prompt

  echo "### Creating uid=proxy account for ${INSTANCE_PREFIX}${1} ###"
  mkdir ${BASE_DIR}/${INSTANCE_PREFIX}${1}/db/proxyUser
  cp ${TEMP_DIR}/proxyUser.ldif ${BASE_DIR}/${INSTANCE_PREFIX}${1}/db/proxyUser

  ${BASE_DIR}/${INSTANCE_PREFIX}${1}/bin/dsconfig \
  create-backend \
   --hostname ${FQDN} \
   --port ${1}444 \
   --bindDn "cn=Directory Manager" \
   --bindPassword password \
   --backend-name proxyUser \
   --type ldif \
   --set enabled:true \
   --set base-dn:uid=proxy \
   --set ldif-file:db/proxyUser/proxyUser.ldif \
   --trustAll \
   --no-prompt

  ${BASE_DIR}/${INSTANCE_PREFIX}${1}/bin/import-ldif \
   --hostname ${FQDN} \
   --port ${1}444 \
   --bindDn "cn=Directory Manager" \
   --bindPassword password \
   --backendId proxyUser \
   --ldifFile ${TEMP_DIR}/proxyUser.ldif \
   --trustAll

  echo "### Adding ACI for proxy account on ${INSTANCE_PREFIX}${1} ###"
  ${BASE_DIR}/${INSTANCE_PREFIX}${1}/bin/ldapmodify \
   --hostname ${FQDN} \
   --port ${1}389 \
   --bindDn "cn=Directory Manager" \
   --bindPassword password <<EOF
dn: ou=tokens
changetype: modify
add: aci
```

```
aci: (target="ldap:///ou=tokens") (targetattr ="*")
 (version 3.0; acl "Allow proxy user to use proxied authz";
 allow(all, proxy) (userdn = "ldap:///uid=proxy");)
EOF
}

# Configure and initialize replication in the shards.
# $1: source instance number
# $2: target instance number
replicate() {
  echo "### Replicating ${INSTANCE_PREFIX}${1} to ${INSTANCE_PREFIX}${2} ###"
  ${BASE_DIR}/${INSTANCE_PREFIX}${1}/bin/dsreplication \
  configure \
   --adminUID admin \
   --adminPassword password \
   --baseDN ou=tokens \
   --host1 ${FQDN} \
   --port1 ${1}444 \
   --bindDN1 "cn=Directory Manager" \
   --bindPassword1 password \
   --replicationPort1 ${1}989 \
   --host2 ${FQDN} \
   --port2 ${2}444 \
   --bindDN2 "cn=Directory Manager" \
   --bindPassword2 password \
   --replicationPort2 ${2}989 \
   --trustAll \
   --no-prompt

  ${BASE_DIR}/${INSTANCE_PREFIX}${1}/bin/dsreplication \
  initialize \
   --adminUID admin \
   --adminPassword password \
   --baseDN ou=tokens \
   --hostSource ${FQDN} \
   --portSource ${1}444 \
   --hostDestination ${FQDN} \
   --portDestination ${2}444 \
   --trustAll \
   --no-prompt
}

create_ldif

cd ${BASE_DIR}

for i in 5 15 6 16
do
  install ${i}
done

replicate 5 15
replicate 6 16

cd ${CURRENT_DIR}
```

b. Adjust variables as described in the script.

c. Run the script.

After the script finishes successfully, four DS replicas in two separate partitions are running on your computer.

5. Set up the DS proxy:

a. Save a local copy of the script, ,

The script is shown in the following listing:

```
#!/usr/bin/env bash
#
# Copyright 2018-2019 ForgeRock AS. All Rights Reserved
#
# Use of this code requires a commercial software license with ForgeRock AS.
# or with one of its affiliates. All use shall be exclusively subject
# to such license between the licensee and ForgeRock AS.
#

###
# Set up a directory proxy server listening on the typical ports.
# This is intended for evaluation on a single system.
#
# In deployment, this would be a layer of identical proxy servers
# to balance incoming requests.
#
# This server distributes data across two replication shards for CTS.
# Each shard is served by two replicas.
# Each DS directory server replicates within one shard only.
###

###
# Adjust these variables to fit your circumstances:
ZIP=~/Downloads/DS-6.5.6.zip
BASE_DIR=/path/to
FQDN=cts.example.com
TEMP_DIR=/tmp
###

CURRENT_DIR=$(pwd)
INSTANCE_PREFIX=proxy

# Install a single proxy from the .zip distribution.
install() {
  echo "### Installing ${BASE_DIR}/${INSTANCE_PREFIX} ###"
  unzip -q ${ZIP}
  mv ${BASE_DIR}/opendj ${BASE_DIR}/${INSTANCE_PREFIX}

  # The default proxyRoot is created at setup time, but removed later.
  ${BASE_DIR}/${INSTANCE_PREFIX}/setup \
  proxy-server \
   --rootUserDN "cn=Directory Manager" \
   --rootUserPassword password \
   --hostname ${FQDN} \
   --ldapPort 1389 \
   --adminConnectorPort 4444 \
   --staticPrimaryServer ${FQDN}:5389 \
   --proxyUserBindDN uid=proxy \
   --proxyUserBindPassword password \
   --proxyUsingStartTLS \
   --trustAll \
   --acceptLicense
   # Note: Do not use trustAll/"Blind Trust" for anything other than evaluation.
   # Either set up a trust manager provider or use the "JVM Trust Manager".

   # Allow the CTS administrator to write directory data.
   ${BASE_DIR}/${INSTANCE_PREFIX}/bin/dsconfig \
   create-global-access-control-policy \
    --bindDn "cn=Directory Manager" \
    --bindPassword password \
    --hostname ${FQDN} \
    --type generic \
    --policy-name "CTS administrator write access" \
    --port 4444 \
    --set allowed-attribute:"*" \
    --set permission:write \
    --set user-dn-equal-to:"uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens" \
    --trustAll \
    --no-prompt
}

# Configure distribution to multiple shards.
configure() {
  echo "### Configuring distribution for CTS shards ###"
  ${BASE_DIR}/${INSTANCE_PREFIX}/bin/dsconfig \
   create-service-discovery-mechanism \
   --bindDn "cn=Directory Manager" \
   --bindPassword password \
   --hostname ${FQDN} \
   --port 4444 \
   --mechanism-name "CTS Shard 1" \
   --type replication \
   --set replication-server:${FQDN}:5444 \
   --set bind-dn:"cn=Directory Manager" \
   --set bind-password:password \
   --set use-start-tls:true \
   --set trust-manager-provider:"Blind Trust" \
   --trustAll \
   --no-prompt

  ${BASE_DIR}/${INSTANCE_PREFIX}/bin/dsconfig \
   create-service-discovery-mechanism \
   --bindDn "cn=Directory Manager" \
   --bindPassword password \
   --hostname ${FQDN} \
   --port 4444 \
   --mechanism-name "CTS Shard 2" \
   --type replication \
   --set replication-server:${FQDN}:6444 \
   --set bind-dn:"cn=Directory Manager" \
   --set bind-password:password \
   --set use-start-tls:true \
   --set trust-manager-provider:"Blind Trust" \
   --trustAll \
   --no-prompt
  ${BASE_DIR}/${INSTANCE_PREFIX}/bin/dsconfig \
   create-backend \
   --bindDn "cn=Directory Manager" \
   --bindPassword password \
   --hostname ${FQDN} \
   --port 4444 \
   --backend-name distributeCts \
   --type proxy \
   --set enabled:true \
   --set partition-base-dn:ou=famrecords,ou=openam-session,ou=tokens \
   --set shard:"CTS Shard 1" \
   --set shard:"CTS Shard 2" \
   --set proxy-user-dn:uid=proxy \
   --set proxy-user-password:password \
   --set route-all:true \
   --trustAll \
   --no-prompt
}

delete_unused_backend() {
  echo "### Removing unused proxyRoot backend ###"
  ${BASE_DIR}/${INSTANCE_PREFIX}/bin/dsconfig \
   delete-backend \
   --bindDn "cn=Directory Manager" \
   --bindPassword password \
   --hostname ${FQDN} \
   --port 4444 \
   --backend-name proxyRoot \
   --trustAll \
   --no-prompt
}

cd ${BASE_DIR}
install
configure
delete_unused_backend
cd ${CURRENT_DIR}
```

b. Adjust variables as described in the script.

c. Run the script.

   After the script finishes successfully, the DS proxy server is configured to distribute requests to the two partitions running on your computer.

6. Install the AM server as described in the AM *Installation Guide*.

   You can use the default configuration options during installation.

7. Configure the AM server to use the directory proxy server as described in *Configuring CTS in the AM Administration Console*.

   This example has the following default settings:

   - Connection string: `cts.example.com:1389`

- Root suffix: `ou=famrecords,ou=openam-session,ou=tokens`
- Login ID: `uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens`
- Password: `password`

When you restart AM, it uses the distributed CTS store. As you perform operations in AM that use CTS, such as logging in as a regular user, you can see the token entries through the DS proxy server.

The following example shows two entries written by AM after logging in through the AM console as the `demo` user:

```
$ /path/to/proxy/bin/ldapsearch -p 1389 \
 -D uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens -w password -b ou=tokens \
 "(coreTokenId=*)" coreTokenId
dn: coreTokenId=YTv/oxEhEfXzkvDkb/7FcdxXSBQ=,ou=famrecords,ou=openam-session,ou=tokens
coreTokenId: YTv/oxEhEfXzkvDkb/7FcdxXSBQ=
```

As AM performs CTS-related operations, you start to see the CTS tokens distributed across the two DS partitions. To examine the results, use LDAP port 5389 for CTS partition 1 and 6389 for partition 2.

8. (Optional) Install a second, identically configured AM server, and then try *Testing Session High Availability*.

9. (Optional) After finishing the evaluation, stop the servers and remove the software you installed:

   a. Uninstall the AM server(s).

   b. Tear down the DS servers:

      i. Save a local copy of the script, ,

         The script is shown in the following listing:

```
#!/usr/bin/env bash
#
# Copyright 2018 ForgeRock AS. All Rights Reserved
#
# Use of this code requires a commercial software license with ForgeRock AS.
# or with one of its affiliates. All use shall be exclusively subject
# to such license between the licensee and ForgeRock AS.
#

###
# Stop and remove the proxy and replica servers.
###

###
# Adjust this variable to fit your circumstances:
BASE_DIR=/path/to
###

INSTANCE_PREFIX=ds-rs-

${BASE_DIR}/proxy/bin/stop-ds
rm -rf ${BASE_DIR}/proxy

for i in 5 15 6 16
do
  ${BASE_DIR}/${INSTANCE_PREFIX}${i}/bin/stop-ds
done
rm -rf ${BASE_DIR}/${INSTANCE_PREFIX}*
```

ii. Adjust the variable as described in the script.

iii. Run the script.

After the script finishes successfully, the DS software has been removed from your computer.

Protecting Data Sovereignty

In many countries, how you store and process user accounts and profile information is subject to regulations and restrictions that protect users' privacy. Data sovereignty legislation is beyond the scope of this document. However, DS servers do include features to help you build services in compliance with data sovereignty requirements.

This section describes how to deploy the following features to align with such requirements:

- Data replication
- Subtree replication
- Fractional replication

The deployments patterns described below address questions of data storage. When planning your deployment, also consider how client applications access and process directory data. By correctly configuring access controls, as described in "*Securing Access to Directory Data*" in the *Security Guide*, you can restrict network access by hostname or IP address, but not generally by physical location of a mobile client application, for example.

Consider developing a dedicated service layer to manage policies that define what clients can access and process based on their location. If your deployment calls for more dynamic access management, use DS together with ForgeRock Access Management software.

Data Replication and Data Sovereignty

Data replication is critical to a high-scale, highly available directory service. For deployments where data protection is also critical, you must, however, make sure that you do not replicate data outside locations where you can guarantee compliance with local regulations.

As described in "Deploying Replication", replication messages flow from directory servers through replication servers to other directory servers. Replication messages contain data that has changed, including data governed by privacy regulations:

- For details on replicating data that must not leave a given location, see "Subtree Replication and Data Protection".
- For details on replicating only part of the data set outside a given location, see "Fractional Replication and Data Sovereignty".

Subtree Replication and Data Protection

As described in "Replication Per Suffix" in the *Administration Guide*, the primary unit of replication is the suffix, specified by a base DN. Subtree replication refers to putting different subtrees (branches) in separate backends, and then replicating those subtrees only to specified servers. For example, you can ensure that the data replicates only to locations where you can guarantee compliance with the regulations in force.

For subtree replication, the RDN of the subtree base DN identifies the subtree. This leads to a hierarchical directory layout. The directory service retains the logical view of a flatter layout, because the branches all join at a top-level base DN.

The following example shows an LDIF outline for a directory service with top-level and local backends:

- The `userRoot` backend holds top-level entries, which do not directly reference users in a particular region.
- The `region1` backend holds entries in the `ou=Region 1,dc=example,dc=com` suffix.
- The `region2` backend holds entries in the `ou=Region 2,dc=example,dc=com` suffix.

The example uses nested groups to avoid referencing local accounts at the top level, but still allowing users to belong to top-level groups:

```
# %<--- Start of LDIF for userRoot --->%
# Base entries are stored in the userRoot backend:
dn: dc=example,dc=com                        # Base DN of userRoot backend
...

dn: ou=groups,dc=example,dc=com              # Stored in userRoot backend
...

dn: ou=Top-level Group,ou=groups,dc=example,dc=com
...
member: ou=R1 Group,ou=groups,ou=Region 1,dc=example,dc=com
member: ou=R2 Group,ou=groups,ou=Region 2,dc=example,dc=com

dn: ou=people,dc=example,dc=com              # Stored in userRoot backend
...

# %<--- End of LDIF for userRoot --->%
# %<--- Start of LDIF for Region 1 --->%
# Subtree entries are stored in a country or region-specific backend.
dn: ou=Region 1,dc=example,dc=com            # Base DN of region1 backend
...

dn: ou=groups,ou=Region 1,dc=example,dc=com   # Stored in region1 backend
...

dn: ou=R1 Group,ou=groups,ou=Region 1,dc=example,dc=com
...
member: uid=aqeprfEUXIEuMa7M,ou=people,ou=Region 1,dc=example,dc=com
...

dn: ou=people,ou=Region 1,dc=example,dc=com   # Stored in region1 backend
...

dn: uid=aqeprfEUXIEuMa7M,ou=people,ou=Region 1,dc=example,dc=com
uid: aqeprfEUXIEuMa7M
...

# %<--- End of LDIF for Region 1 --->%
# %<--- Start of LDIF for Region 2 --->%
dn: ou=Region 2,dc=example,dc=com            # Base DN of region2 backend
...

dn: ou=groups,ou=Region 2,dc=example,dc=com   # Stored in region2 backend
...

dn: ou=groups,ou=R2 Group,ou=Region 2,dc=example,dc=com
...
member: uid=8EvlfE0rRa3rgbX0,ou=people,ou=Region 2,dc=example,dc=com
...

dn: ou=people,ou=Region 2,dc=example,dc=com   # Stored in region2 backend
...

dn: uid=8EvlfE0rRa3rgbX0,ou=people,ou=Region 2,dc=example,dc=com
uid: 8EvlfE0rRa3rgbX0
...
```

```
# %<--- End of LDIF for Region 2 --->%
```

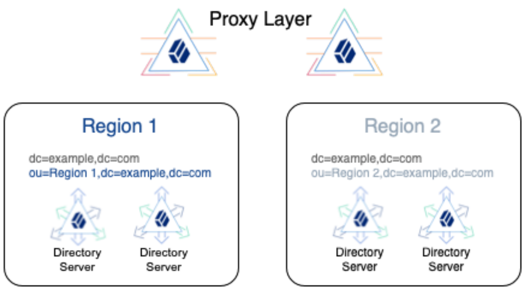
The deployment for this example has the following characteristics:

- The LDIF is split at the comments about where to cut the file — `# %<--- Start|End of LDIF for ... --->%`.
- All locations share the LDIF for `dc=example,dc=com`, but the data is not replicated.

  If DS replicates `dc=example,dc=com`, it replicates all data for that base DN, which includes *all* the data from *all* regions.

  Instead, minimize the shared entries, and manually synchronize changes across all locations.
- The local LDIF files are constituted and managed only in their regions:
  - Region 1 data is only replicated to servers in region 1.
  - Region 2 data is only replicated to servers in region 2.
- The directory service only processes information for users in their locations according to local regulations.

Separate Replication Domains for Data Sovereignty

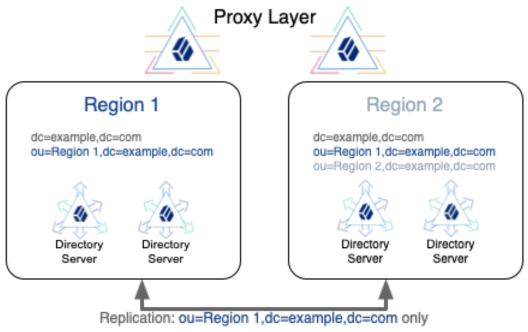


In a variation on the deployment shown above, consider a deployment with the following constraints:

- Region 1 regulations allow region 1 user data to be replicated to region 2.

  You choose to replicate the region 1 suffix in both regions for availability.
- Region 2 regulations do not allow region 2 user data to be replicated to region 1.

Mixed Replication Domains for Data Sovereignty


Proxy Layer
Region 1
dc=example,dc=com
ou=Region 1,dc=example,dc=com
Directory Server
Directory Server
Region 2
dc=example,dc=com
ou=Region 1,dc=example,dc=com
ou=Region 2,dc=example,dc=com
Directory Server
Directory Server
Replication: ou=Region 1,dc=example,dc=com only

When you use subtree replication in this way, client applications can continue to read and update directory data as they normally would. Directory servers only return data that is locally available.

For additional information, see "Subtree Replication" in the *Administration Guide*, and "Splitting Data Across Multiple Backends" in the *Administration Guide*.

Fractional Replication and Data Sovereignty

In some deployments, regulations allow you to replicate some user attributes. For example, consider a deployment where data sovereignty regulations in one region allow UIDs and class of service levels to be present everywhere, but do not allow users' personal information to leave their location.

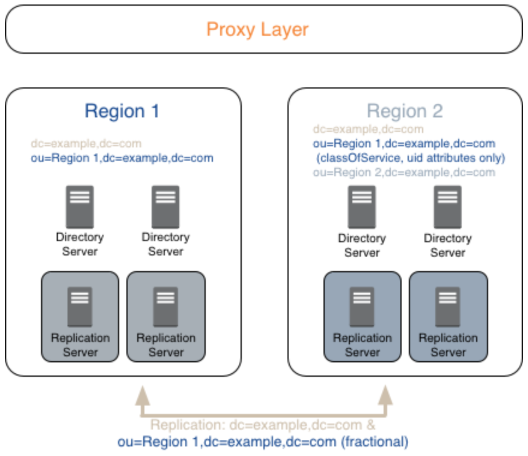
Consider the following entry where the attribute values are preceded by the comment, `# Can be replicated everywhere`, are the values that you can replicate outside the user's region:

```
dn: uid=aqeprfEUXIEuMa7M,ou=people,ou=Region 1,dc=example,dc=com
objectClass: top
objectClass: cos
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
# Can be replicated everywhere:
classOfService: bronze
cn: Babs Jensen
cn: Barbara Jensen
facsimiletelephonenumber: +1 408 555 1992
gidNumber: 1000
givenname: Barbara
homeDirectory: /home/bjensen
l: Region 1
mail: bjensen@example.com
manager: uid=2jD5NanzOZGjMmcz,ou=people,ou=Region 1,dc=example,dc=com
ou: People
ou: Product Development
preferredLanguage: en, ko;q=0.8
roomnumber: 0209
sn: Jensen
telephonenumber: +1 408 555 1862
# Can be replicated everywhere:
uid: aqeprfEUXIEuMa7M
uidNumber: 1076
userpassword: {SSHA512}2olpp6P5dWDZyYmOQwyk...9fOSlN4ZrpJcrUz2qVMiRpplkI
```

To allow you to replicate only a portion of each entry, DS servers implement fractional replication. You configure fractional replication by updating the directory server configuration to specify which attributes to include or exclude in change messages from replication servers to the directory server replica.

The replication server must remain located with the directory server replicas that hold full entries which include all attributes. The replication server can receive updates from these replicas, and also from replicas that hold fractional entries. Each replication server must therefore remain within the location where the full entries are processed. Otherwise, replication messages describing changes to protected attributes would be sent outside the location where the full entries are processed.

Fractional Replication for Protected Data


Proxy Layer
Region 1
dc=example,dc=com
ou=Region 1,dc=example,dc=com
Directory Server
Directory Server
Replication Server
Replication Server
Region 2
dc=example,dc=com
ou=Region 1,dc=example,dc=com
(classOfService, uid attributes only)
ou=Region 2,dc=example,dc=com
Directory Server
Directory Server
Replication Server
Replication Server
Replication: dc=example,dc=com &
ou=Region 1,dc=example,dc=com (fractional)

To leave schema checking enabled on the replicas that receive fractional updates, portions of entries that are replicated must themselves be complete entries. In other words, in the example above, the entry's structural object class would have to allow `classOfService` and `uid`. This would require editing the schema and the `objectClass` values of the entries. For details, see "*Managing Schema*" in the *Administration Guide*.

For additional information, see "Fractional Replication" in the *Administration Guide*.

Integrating With Non-ForgeRock Directories

This section covers some common use cases of interoperability with other directory services and directory client applications.

Use Case	**See...**
More than one directory service	"Using a Proxy Layer"
Credentials in another directory service	"Using Another Service for Authentication"
Must sync changes across directory services	"Implementing Directory Data Synchronization or Data Migration"
Web clients need alternate data views	"Providing Alternate Views of Directory Data"

Using a Proxy Layer

Adding a directory proxy layer can help you deploy alongside an existing directory service. The proxy layer lets you provide a single entry point to both new and old directory services.

You configure a directory proxy server to connect to servers in each directory. DS proxy servers can discover DS directory servers by connecting to DS replication servers. For other directories, you must statically enumerate the directory server to contact. DS proxy servers work with any LDAP directory server that supports the standard proxied authorization control defined in RFC 4370.

Each DS proxy server forwards client requests to the directory service based on the target DN of the operation. As long as the base DNs for each directory service differ, the proxy layer can provide a single entry point to multiple directory services.

For details, see "*Configuring LDAP Proxy Services*" in the *Administration Guide*, and in particular "Deploying a Single Point of Directory Access" in the *Administration Guide*.

Using Another Service for Authentication

For cases where an existing directory service holds authentication credentials, DS servers provide a feature called pass-through authentication.

With pass-through authentication, the DS server effectively redirects LDAP bind operations to a remote LDAP directory service. If the DS and remote user accounts do not have the same DN, you configure the DS server to automatically map local entries to the remote entries. Pass-through authentication can cache passwords if necessary for higher performance with frequent authentication.

For details, see "*Configuring Pass-Through Authentication*" in the *Administration Guide*.

Implementing Directory Data Synchronization or Data Migration

You may need to continually synchronize changes across multiple services, or to migrate data from an existing directory service.

For ongoing data synchronization across multiple services, consider ForgeRock Identity Management software or a similar solution. ForgeRock Identity Management software supports configurable data reconciliation and synchronization at high scale and with multiple data sources, including directory services.

For one-time upgrade and data migration to DS software, the appropriate upgrade and migration depends on your deployment:

- **Offline Migration**

  When downtime is acceptable, you can synchronize data, then migrate applications to the DS service and retire the old service.

  Depending on the volume of data, you might export LDIF from the old service and import LDIF into the DS service during the downtime period. In this case, stop the old service at the beginning of the downtime period to avoid losing changes.

If the old service has too much data to fit the export/import operation into the downtime period, you can perform an export/import operation before the downtime starts, but you must then implement ongoing data synchronization from the old service to the DS service. Assuming you can keep the new DS service updated with the latest changes, the DS service will be ready to use. You can stop the old service after migrating the last client application.

- **Online Migration**

  When downtime is not acceptable, both services continue running concurrently. You must be able to synchronize data, possibly in both directions. ForgeRock Identity Management software supports bi-directional data synchronization.

  Once you have bi-directional synchronization operating correctly, migrate applications from the old service to the DS service. You can stop the old service after migrating the last client application.

Providing Alternate Views of Directory Data

Not all directory clients expect the same directory data. Clients might even expect completely different identity objects.

DS servers expose the same LDAP data view to all directory clients. (You can adjust this behavior somewhat for update operations as described in "Filtering Add and Modify Operations" in the *Developer's Guide*.)

The RESTful views of directory data for HTTP clients are fully configurable, however. By developing alternative REST to LDAP mappings and exposing multiple APIs, or different versions of the same API, you can present directory data in different ways to different applications. For details, see "To Set Up REST Access to User Data" in the *Administration Guide*, and "*REST to LDAP Configuration*" in the *Reference*.

Chapter 5
Provisioning Systems

This chapter covers hardware and software prerequisites for your deployment.

Sizing Systems

Given availability requirements and estimates on sizing for services, estimate the required capacity for individual systems, networks, and storage. This section considers only systems for DS servers, not monitoring and audit tools, backup storage, or client applications.

CPU, memory, network, and storage requirements depend in large part on the services you plan to provide. The indications in "Choosing Hardware" are only starting points for your sizing investigation.

For details about how each component uses system resources, see "*Understanding Directory Components*".

Sizing System CPU

Directory servers consume significant CPU resources when processing username-password authentications where the password storage scheme is computationally intensive (Bcrypt, PBKDF2, PKCS5S2).

Warning

Using a computationally intensive password storage scheme such as Bcrypt will have a severe impact on performance. Before you deploy a computationally intensive password storage scheme in production, you *must* complete sufficient performance testing and size your deployment appropriately. Provision enough CPU resources to keep pace with the peak rate of simple binds. If you do not complete this testing and sizing prior to deploying in production, you run the risk of production outages due to insufficient resources.

DS servers also use CPU resources to decode requests and encode responses, and to set up secure connections. LDAP is a connection-oriented protocol, so the cost of setting up a connection may be small compared to the lifetime of the connection. HTTP, however, requires a new connection for each operation. If you have a significant volume of HTTPS traffic, provision enough CPU resources to set up secure connections.

Sizing System Memory

Directory server memory requirements depend primarily on how you cache directory data. Be aware that default settings cause the directory server to request half the available memory for each directory database backend. At minimum, therefore, you must revisit this setting if you have more than one directory database backend. If your directory data set can fit entirely into system memory and you want to optimize for fast reads and searches, provision enough RAM to cache everything.

If caching the entire directory data set is not an option, provision enough RAM to cache at least all internal database nodes. For details, see "Database Cache Settings" in the *Administration Guide*.

DS servers also use memory to maintain active connections and processes. As indicated in "Fulfilling Memory Requirements", provision an additional minimum of 2 GB RAM or more depending on the volume of traffic to your service.

Sizing Network Connections

When sizing network connections, account for all requests and responses, including replication traffic. When calculating request and response traffic, base your estimates on your key client applications. When calculating replication traffic, be aware that all write operations must be communicated over the network, and replayed on each directory server. Each write operation results in at least N-1 replication messages, where N is the total number of servers. Also, keep in mind that all DS servers running a replication service are fully connected, including those servers that are separated by WAN links.

For deployments in multiple regions, account especially for traffic over WAN links, as this is much more likely to be an issue than traffic over LAN links.

Make sure to size enough bandwidth for peak throughput, and do not forget redundancy for availability.

Sizing Disk I/O and Storage

The largest disk I/O loads for DS servers arise from logging and writing directory data. You can also expect high disk I/O when performing a backup operation or exporting data to LDIF.

I/O rates depend on the service levels that the deployment provides. When you size disk I/O and disk space, you must account for peak rates and leave a safety margin when you must briefly enable debug logging to troubleshoot any issues that arise.

Also, keep in mind the possible sudden I/O increases that can arise in a highly available service when one server fails and other servers must take over for the failed server temporarily.

DS server access log files grow more quickly than other logs. Default settings prevent each access logger's files from growing larger than 2 GB before removing the oldest. If you configure multiple access loggers at once, multiply 2 GB by their number.

Directory server database backend size grows as client applications modify directory data. Even if data set's size remains constant, the size of the backend grows. Historical data on modified directory entries increases until purged by the directory server when it reaches the replication purge delay (default: 3 days). In order to get an accurate disk space estimate, follow the process described in "Planning for High Scale".

Replication server changelog backend size is subject to the same growth pattern as historical data. Run the service under load until it reaches the replication purge delay to estimate disk use.

For highest performance, use fast SSD disk and separate disk subsystems logging, backup, and database backends.

Concerning Portability

DS client and server code is pure Java, and depends only on the JVM. This means you can run clients and servers on different operating systems, and copy backup files and archives from one system to another.

Server Portability

DS servers and data formats are portable across operating systems. When using multiple operating systems, nevertheless take the following features into account:

Command-Line Tool Locations

DS server and command-line tools are implemented as scripts. The path to the scripts differ on UNIX/Linux and Windows systems. Find UNIX/Linux scripts in the `bin` directory. Find Windows scripts in the `bat` folder.

Native Packaging

When you download DS server software, you choose between cross-platform and native packages.

- Cross-platform .zip packaging facilitates independence from the operating system. You manage the server software in the same way, regardless of the operating system.
- Native packaging facilitates integration with the operating system. You use the operating system tools to manage the software.

Both packaging formats provide scripts to help register the server as a service of the operating system. These scripts are **create-rc-script** (UNIX/Linux) and **windows-service** (Windows).

Gateway Portability

The DS DSML and REST to LDAP gateway web applications run on any supported application server, regardless of the underlying operating system.

The only persistent state for gateway applications is in their configuration files. The gateway configuration files are portable across application servers and operating systems.

Choosing Hardware

Thanks to the underlying Java platform, Directory Services software runs well on a variety of processor architectures. Many directory service deployments meet their service-level agreements without the very latest or very fastest hardware.

Fulfilling Memory Requirements

When installing a directory server for evaluation, you need 256 MB memory (32-bit) or 1 GB memory (64-bit) available.

For installation in production, read the rest of this section. You need at least 2 GB memory for a directory server and four times the disk space needed for initial production data in LDIF format. A replicated directory server stores data, indexes for the data, operational attribute data, and historical information for replication. The server configuration trades disk space for performance and resilience, compacting and purging data for good performance and for protection against temporary outages. In addition, leave space for growth in database size as client applications modify and add entries over time.

For a more accurate estimate of the disk space needed, import a known fraction of the initial LDIF with the server configured for production. Run tests to estimate change and growth in directory data, and extrapolate from the actual space occupied in testing to estimate the disk space required in production.

Directory servers almost always benefit from caching all directory database files in system memory. Reading from and writing to memory is much faster than reading from and writing to disk storage.

For large directories with millions of user directory entries, there might not be room to install enough memory to cache everything. To improve performance in such cases, use quality solid state drives either for all directory data, or as an intermediate cache between memory and disk storage.

Fulfilling Minimum Disk Space Requirements

To evaluate DS software, make sure you have 10 GB free disk space for the software and for sample data.

The more data you have, the more disk space you need. Before deploying production systems, make sure you have enough space. For details, see "Planning for High Scale".

Choosing a Processor Architecture

Processor architectures that provide fast single thread execution tend to help Directory Services software deliver the lowest response times. For top-end performance in terms of sub-millisecond response times and of throughput ranging from tens of thousands to hundreds of thousands of operations per second, the latest x86/x64 architecture chips tend to perform better than others.

When deploying DS servers with replication enabled, allow at minimum two CPU cores per server. Allow more CPU cores per server, especially in high-volume deployments or when using CPU-intensive features such as encryption. Single CPU systems seriously limit server performance.

Chip multi-threading (CMT) processors can work well for directory servers providing pure search throughput, though response times are higher. However, CMT processors are slow to absorb hundreds or thousands of write operations per second. Their slower threads get blocked waiting on resources, and thus are not optimal for deployments with high write throughput requirements.

Fulfilling Network Requirements

On systems with fast processors and enough memory to cache directory data completely, the network can become a bottleneck. Even if a single 1 Gb Ethernet interface offers plenty of bandwidth to handle your average traffic load, it can be too small for peak traffic loads. Consider using separate interfaces for administrative traffic and for application traffic.

To estimate the network hardware required, calculate the size of the data returned to applications during peak load. For example, if you expect to have a peak load of 100,000 searches per second, each returning a full 8 KB entry, you require a network that can handle 800 MB/sec (3.2 Gb/sec) throughput, not counting other operations, such as replication traffic.

Fulfilling Storage Requirements

Warning

The directory server does not currently support network file systems such as NFS for database storage. Provide sufficient disk space on local storage such as internal disk or an attached disk array.

For a directory server, storage hardware must house both directory data, including historical data for replication, and server logs. On a heavily used server, you might improve performance by putting access logs on dedicated storage.

Storage must keep pace with throughput for write operations. Write throughput can arise from modify, modify DN, add, and delete operations, and from bind operations when a login timestamp is recorded, and when account lockout is configured, for example.

In a replicated topology, a directory server writes entries to disk when they are changed, and a replication server writes changelog entries. The server also records historical information to resolve potential replication conflicts.

As for network throughput, base storage throughput required on peak loads rather than average loads.

Choosing an Operating System

Directory Services 6.5 software is supported on the following operating systems:

- Linux 2.6 and later
- Microsoft Windows Server 2008, 2008 R2, 2012, 2012 R2, and 2016
- Oracle Solaris 10, 11 (SPARC, x64)

In order 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 how to set the journaling mode for data, see the options for your file system in the **mount** command manual page.

Setting Maximum Open Files

DS servers need to be able to open many file descriptors, especially when handling thousands of client connections. Linux systems in particular often set a limit of 1024 per user, which is too low to handle many client connections to the DS server.

When setting up your DS server for production use, 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, you can 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. You can check the Linux system overall maximum as follows:

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

Setting Maximum Inotify Watches

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, change it as shown in the following example:

```
$ sudo sysctl --write fs.inotify.max_user_watches=524288
[sudo] password for admin:
fs.inotify.max_user_watches = 524288
```

Preventing Interference With Antivirus Software

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 database file access.

Antivirus and intrusion detection software can interfere with the normal process of opening and closing database working files. They may incorrectly mark such files as suspect to infection due to normal database processing, which involves opening and closing files in line with the database's internal logic.

Prevent antivirus and intrusion detection systems from scanning database and changelog database files.

At minimum, configure antivirus software to whitelist the DS server database files. By default, exclude the following file system directories from virus scanning:

- `/path/to/opendj/changelogDb/` (if replication is enabled)

  Prevent the antivirus software from scanning these changelog database files.

- `/path/to/opendj/db/`

  Prevent the antivirus software from scanning database files, especially `*.jdb` files.

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 "Limiting System and Administrative Access" in the *Security Guide*.

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.

Preparing the Java Environment

Directory Services software consists of pure Java applications. Directory Services servers and clients run on any system with full Java support. Directory Services is tested on a variety of operating systems, and supported on those listed in "Choosing an Operating System".

Directory Services 6.5 software requires Java 8 or 11, specifically at least the Java Standard Edition runtime environment, or the corresponding Java Development Kit to compile Java plugins and applications.

Note

ForgeRock validates Directory Services software with OpenJDK and Oracle JDK, and does occasionally run sanity tests with other JDKs such as the IBM JDK and Azul's Zulu. Support for very specific Java and hardware combinations is best-effort. This means that if you encounter an issue when using a particular JVM/hardware combination, you must also demonstrate the problem on a system that is widespread and easily tested by any member of the community.

ForgeRock recommends that you keep your Java installation up-to-date with the latest security fixes.

Important

Directory server JE database backends can require additional JVM options. When running a directory server with a 64-bit JVM and less than 32 GB maximum heap size, you must use the Java option, `-XX:+UseCompressedOops`. To use the option, edit the `config/java.properties` file. The following example settings include the option with the arguments for offline LDIF import, for rebuilding backend indexes, and for starting the directory server:

```
import-ldif.offline.java-args=-server -XX:+UseCompressedOops
rebuild-index.offline.java-args=-server -XX:+UseCompressedOops
start-ds.java-args=-server -XX:+UseCompressedOops
```

Make sure you have a required Java environment installed on the system. If your default Java environment is not appropriate, set `OPENDJ_JAVA_HOME` to the path to the correct Java environment, or set `OPENDJ_JAVA_BIN` to the absolute path of the **java** command. The `OPENDJ_JAVA_BIN` environment variable is useful if you have both 32-bit and 64-bit versions of the Java environment installed, and want to make sure you use the 64-bit version.

Running in a Container

For some settings, DS servers depend on system information reported by the JVM to determine defaults. When running DS servers in containers such as Docker, the Java 8 JVM returns information about the operating system that does not reflect container constraints and limits. When using Java 8, manually adjust the settings described below.

Note

Java 11 supports gathering container information, as described in JDK-8146115. This fix was backported to Java 8 update 191, as mentioned in the JDK 8u191 Update Release Notes.

Skip this section when using Java 8 update 191 or later, or Java 11.

If necessary, override automatic CPU detection by specifying the number of CPUs the JVM uses with `-XX:ActiveProcessorCount=`*`count`* in `config/java.properties`.

Before adjusting settings, determine the following container constraints:

- The number of CPU core hardware threads dedicated to the containerized system, which is usually twice the number of CPU cores
- The amount of RAM dedicated to the containerized system

When running DS servers in containers such as Docker, adjust the following settings:

- `num-request-handlers`

  Recommendation: Set this either to 2 or to 1/4 of the number of core hardware threads, whichever is larger.

- `num-worker-threads`

  Recommendation: Set this either to 4 or to 5/8 of the number of core hardware threads, whichever is larger.

- `db-num-cleaner-threads`

  Recommendation: Set this either to 2 or to 1/4 of the number of core hardware threads, whichever is larger.

- `num-update-replay-threads`

  Recommendation: Set this either to 4 or to 1/2 of the number of core hardware threads, whichever is larger.

- `-Xmx` (Java setting limiting maximum heap size)

  To use the option, edit the `config/java.properties` file and restart the server.

  For example, consider a container limited to 8 GB RAM. The following setting limits the maximum heap size to 8 GB when starting the directory server:

  ```
  start-ds.java-args=-server -Xmx8G
  ```

Choosing an Application Server

DS servers run as standalone Java services, and do not depend on an application server.

The REST to LDAP and DSML gateway applications run on Apache Tomcat (Tomcat) and Jetty.

ForgeRock supports only stable application container releases. See the Tomcat and Jetty documentation for details about the right container to use with your Java environment.

Assigning FQDNs For Replication

Directory Services replication requires use of fully qualified domain names (FQDNs), such as `opendj.example.com`.

Host names like `my-laptop.local` are acceptable for evaluation. In production, and when using replication across systems, you must either ensure DNS is set up correctly to provide FQDNs, or update the hosts file (`/etc/hosts` or `C:\Windows\System32\drivers\etc\hosts`) to supply unique, FQDNs.

Synchronizing System Clocks For Replication

When using DS replication, keep server system clocks synchronized.

To keep the system clocks synchronized, use a tool that always moves the clock forwards. For example, `ntpd` adjusts the size of a second so that time always moves forwards to eventual clock consistency.

Never move the system clock *backwards*. Never use tools such as **ntpdate** that may move the clock backwards.

Getting Digital Certificates Signed

If you plan to configure SSL or TLS to secure network communications between the server and client applications, install a properly signed digital certificate that your client applications recognize, such as one that works with your organization's PKI or one signed by a recognized certificate authority.

To use the certificate during installation, the certificate must be located in a file-based keystore supported by the JVM (JKS, JCEKS, PKCS#12), or on a PKCS#11 token. To import a signed certificate into a keystore, use the Java **keytool** command.

For details, see "Preparing For Secure Communications" in the *Administration Guide*.

Chapter 6

Deploying for DevOps and SaaS

This chapter focuses on use of DS software in DevOps and SaaS deployments.

Tip

ForgeRock publishes example code demonstrating how to build Docker images, and how to run components of the ForgeRock platform together in Kubernetes-based deployments. For details, see the ForgeRock *DevOps Documentation*.

The points and recommendations made in this chapter apply to any deployment that relies heavily on automation. This chapter is not tied to any specific DevOps software.

> Twelve-factor processes are stateless and share-nothing.
>
> — The Twelve-Factor App: Processes

The twelve-factor methodology focuses on building software-as-a-service (SaaS) applications that work well with DevOps processes and run smoothly in cloud environments. The methodology recommends combining twelve-factor applications with backing services for data and other requirements.

The directory provides a backing service. It is at odds with some twelve-factor propositions, such as being stateless. That does not prevent you from using DS software to build a SaaS application, or from using DS software in a DevOps or cloud environment. It does mean that you must manage DS application data, which is the state that stateless applications avoid.

In a DevOps environment, you distinguish between *immutable* and *mutable* states. Plan to change immutable state only in your development environment. Plan to maintain mutable state separately for each environment.

The following list distinguishes immutable and mutable state in the context of DS servers:

- Immutable software

  This includes top-level files and the contents of the following directories: `bat/`, `bin/`, `classes/`, `extlib/`, `legal-notices/`, `lib/`, `snmp/`, `template/`.

- Immutable configuration data

  This includes the contents of the following directories: `config/`, potentially `db/schema/`.

- Mutable configuration data

This includes the contents of the following directories: `db/adminRoot/`, `db/ads-truststore/`, `db/monitorUser/`, `db/rootUser/`, `db/tasks/`, `var/`.

- Mutable application data

  This includes the contents of the `db/userRoot/` directory, and directories for other backends that you create.

- Mutable server data

  This includes the contents of the following directories: `locks/`, `logs/`.

- Mutable replication changelog data

  This includes the contents of the `changelogDb/` directory.

- Mutable backup archives and LDIF

  This includes the contents of the following directories: `bak/`, `import-tmp/`, `ldif/`.

These distinctions apply to directory servers, replication servers, and directory proxy servers, although not all mutable state is present on each type of server.

The DSML and REST to LDAP gateways have no state of their own, only configuration. As the gateway configuration files do not natively support expressions and configuration from the environment, you must manage the configuration properties that identify backing services, such as configuration for connections to LDAP directory services.

Managing Immutable Software

With the DS `.zip` distribution, all software files unpack into an `opendj` directory. The only direct dependency is on a supported version of the Java platform.

Do not move or alter files delivered with the distribution. Add your customizations as additional files. As described in "Developing Custom Server Plugins", be aware that you may need to recompile and potentially update extensions for each new DS software release. Consider each custom extension as sub-project with its own acceptance criteria.

As described in "Upgrading", the upgrade process consists of unpacking the software over the existing files, overwriting old files, and then running the **upgrade** command. The **upgrade** command adapts the configuration to work with the new software binaries. This process will break if you move or alter files delivered with the distribution.

Managing Immutable Configuration State

Immutable configuration state does not change between development, test, and production environments.

These recommendations concern immutable configuration files:

`config/config.ldif`

The main DS server configuration, stored in the `config/config.ldif` file, supports configuration expressions substituted with values from the environment at server startup. As you develop your configuration, you replace static values in the configuration with expressions. At runtime, you provide values for these expressions through environment variables, system properties, or separate configuration files. The server replaces the expressions with their values at startup time.

For examples, see "Using Server Configuration Expressions". For details, see "Using Configuration Property Value Substitution" in the *Administration Guide*.

The supported interface to configure servers is the command-line tools, such as the **dsconfig** command. Although it is possible, for example, to modify the `config/config.ldif` file directly when the server is not running, there is no guarantee that the LDIF representation will remain the same from one version to another. The file format is subject to change without warning, whereas the command interface is stable.

`config/`

All the other files under the `config/` directory can be immutable at runtime, assuming the following:

- The server obtains access to public and private keys from the environment at runtime.
- The server obtains configuration files for any ForgeRock Common Audit external log publishers, such as those for Elasticsearch, a JDBC database, JMS, Splunk, or Syslog, from the environment at runtime.

`db/schema/`

LDAP schemas can be updated at runtime, and are replicated by default. In a DevOps environment, record tested schema changes in the development environment.

To allow testing of new schema definitions in a test or production environment, set the `enabled` property for schema replication with a configuration expression, as described in "Using Server Configuration Expressions". You can then use an environment setting to turn schema replication off before testing new definitions on a specified server.

Be aware that the files under the `db/schema/` directory include the schema for the LDAP representation of server configuration, and for standard attributes and object classes. Do not modify the schema delivered with the software, but instead add your own definitions as described in "*Managing Schema*" in the *Administration Guide*.

Managing Mutable State

Mutable state is different in development, test, and production environments.

Once you have initialized mutable state in an environment, that state has its own lifecycle. Mutable state must be persisted in that environment.

When you first deploy into an environment, copy the mutable state from your development environment into the persistent location. Once initialized, mount the persistent mutable state from the environment, and use it instead.

Mutable Configuration

`db/adminRoot/`
`db/ads-truststore/`
`db/tasks/`
`var/`

Let the server manage these. Do not update them directly.

`db/monitorUser/`
`db/rootUser/`

The administrative user entries are in LDIF backends by default.

Make sure the authentication credentials are appropriate for each environment. At minimum, use strong passwords in your production environment. When updating the password in an LDIF backend, use either of the following mechanisms:

- Replace the `userPassword` value in the LDIF with the new cleartext password value, and then import the modified LDIF into the backend.
- Replace the `userPassword` value directly in the LDIF with the encoded output of the **encode-password** command. Do this while the server is stopped.
- If these have been converted to database backends, use the **ldappasswordmodify** command.

Mutable Application Data

`db/userRoot/`
Any additional backends added in your deployment

In high-volume environments, keep these backends on fast disk with high throughput and low latency.

If you use data confidentiality (encryption) or reversible passwords such as AES, be aware that your application data has a strong link with the contents of the `db/adminRoot/` and `db/ads-truststore/` directories. As described in "Cryptographic Key Management" in the *Security Guide*, the symmetric key is stored in the `adminRoot` backend, encrypted with the public key of the key pair stored in the `ads-truststore` backend.

Mutable Server Data

`locks/`

Let the server manage these when it is running. You can safely delete the `*.lock` files when the server is not running.

`logs/`

> A twelve-factor app ... should not attempt to write to or manage logfiles.
>
> — The Twelve-Factor App: Logs

By default, DS servers write different information to different logs, as described in "Server Logs" in the *Administration Guide*:

- Access logs for messages about clients accessing the server.

  You can configure the server to send access log messages to an external service, such as Elasticsearch, a JDBC database, JMS, Splunk, or Syslog. Access logs generally consume more disk IO and space, by far, than other other logs, since debug logs are unused by default. These are the logs to analyze for understanding how clients use your directory service.

- Debug logs for messages tracing internal server events.

  Turn on debug logging only for troubleshooting.

- Error logs for messages tracing server events.
- Replication logs for messages used to help repair problems in data replication.
- A `server.out` log for messages about server events since startup.

  Messages in this file have the same format as error log messages.

You can configure DS servers to write access and `server.out` log messages to standard output. To do so, perform the following high-level steps:

1. Configure the server as described in "Sending JSON Access Logs to Standard Output" in the *Administration Guide*.
2. Disable access log publishers that write messages to files you do not use.
3. Start the server with the **start-ds --noDetach** command.

   The `--noDetach` option causes the server to run in the foreground, rather than detaching from the terminal and running as a daemon.

Because logs are indispensable for troubleshooting, you may choose to retain them separately from the server instance. By default, DS servers rotate log files and retain log files only until the cumulative size crosses a configurable threshold. Beyond the threshold, DS servers remove the oldest log files. The default threshold for access log files is 2 GB.

In high-volume environments, keep the logs on fast disk with high throughput and low latency.

Mutable Replication Changelog Data

`changelogDb/`

Let the server manage changelog data. The server purges this data periodically depending on your replication purge delay settings.

In high-volume environments, keep changelog data on fast disk with high throughput and low latency.

Mutable Backup Archives and LDIF Files

`bak/`

This is the default location where the backup command stores backup archives. This is generally mounted on large, cheaper disks.

`import-tmp/`

This is the default working directory used when importing LDIF data.

`ldif/`

This is the default location for saving exported LDIF files. This is generally mounted on large, cheaper disks.

Working in a Development Environment

The following high-level stages outline the process for preparing a directory server replica with its own changelog service in your development environment:

1. Download the DS software distribution.
2. Install two directory servers.
3. Configure the servers as replicas of each other as described in "Configuring Replication Settings" in the *Administration Guide*.
4. Stop the servers.
5. Remove the second server and delete it.
6. Replace environment-dependent values in the first server's configuration, making it a template for further development.
7. Apply additional changes to the template server's configuration as necessary.

For a directory proxy server, skip the stages pertaining to replication configuration. Install only one directory proxy server before using expressions in its configuration.

Unless you have a global scenario with multiple regions, use combined directory server/replication server instances instead of standalone directory servers and standalone replication servers. This reduces the number of server templates to manage.

Installing Directory Replicas

The data in different DevOps environments will not be the same. For example, in the development environment, you are likely to use only a few generated entries in each backend. In the production environment, you might have millions of entries subject to live updates at any time in backends that you maintain for years.

When installing directory replicas, create a separate backend for each base DN you will use in production. Also, create at least a base entry at setup time for each backend.

Important

DS replication works by sharing changes rather than sharing data. When an update occurs, the replica making the update shares the update with all other replicas through the replication protocol. Each replica then converges on the same state by applying local updates and updates from other replicas in accordance with rules of conflict resolution. As long as each replica starts from the same initial state, each replica eventually ends up in the same convergent state. It is therefore crucial that each replica begin with the same initial state. There would be no way to converge from different initial states.

Internally, DS replicas store a shorthand form of the initial state called a generation ID. A generation ID is a hash of the first 1000 entries in a backend. Replication uses the generation ID at startup time to determine whether the current initial state of the backend corresponds to the known initial state for replication. If the two generation IDs do not match for replicated backends, then you must intervene to initialize replication to the same state on all replicas.

The generation ID is stored both in the root entry of a backend on the `ds-sync-generation-id` operational attribute, and in a file in the changelog database.

To transform the initial server into a template for further development, you will prepare an immutable configuration. You create at least the base entry at setup time for each backend, so that there is somewhere in the backend to store the generation ID.

When a combined directory server/replication server starts up, if the generation ID file in the changelog is missing, the replication server can recover the generation ID in the changelog database from the generation ID in the local backend files. In this case, you must then carefully ensure that all replicas start with exactly the same initial data as described below. This allows you to remove the changelog data—which differs in each environment—from your template replica without breaking replication.

Be aware that this recommendation applies for the current release. The generation ID is an internal mechanism, not part of any public or stable interface. This could change in a future release.

The setup process requires a keystore holding public key certificates and private keys for secure communications with other applications. In a DevOps deployment, the default is to generate a file-based keystore that contains public and private keys. You should instead install the server with your own keystore or other method of accessing keys.

The setup process also creates a file-based keystore, `db/ads-truststore`, to hold the keys for replication and symmetric key distribution. Let the server manage this keystore, and make it part of your template replica image. Do not replace it with settings from the environment. The mechanism for allowing you to manage these keys as part of the DevOps environment is evolving.

Setting Up Replication

While preparing DS replicas in your development environment, configure the servers as replicas of each other as described in "Configuring Replication Settings" in the *Administration Guide*.

Some subcommands of the **dsreplication** command, such as **status**, are not compatible with configuration expressions. Do not use the **dsreplication** command after initial setup. After you use expressions in configuration settings, monitor replication delay over HTTP or LDAP instead.

Using Server Configuration Expressions

After setting up replication, perform these high-level steps:

1. Stop the servers.

2. Remove the changelog database content from the server you will use as a template:

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

   As described above, you will carefully ensure in each environment that all replicas start with exactly the same initial data.

3. Use configuration expressions to template the server's configuration in offline mode. The basic settings are described in "Templating a Server Configuration With Expressions".

   You can make the changes interactively while the server is stopped by running the **dsconfig** command in offline mode with the configuration file option. For example:

   ```
   $ /path/to/opendj/bin/dsconfig --offline --configFile /path/to/opendj/config/config.ldif
   ```

   Alternatively, you can perform the commands above with the **dsconfig** command in offline mode using the batch file option, as in the following example:

```
$ dsconfig \
 --offline \
 --configFile /path/to/opendj/config/config.ldif \
 --batch \
 --no-prompt <<EOF
set-global-configuration-prop --set server-id:\&{ds.server.id}
set-sasl-mechanism-handler-prop --handler-name DIGEST-MD5 --set server-fqdn:\&{ds.fqdn}
set-backend-prop --backend-name userRoot --set enabled:\&{ds.userRoot.enabled\|true}
set-replication-domain-prop --provider-name Multimaster\ Synchronization \
 --domain-name dc=example,dc=com --set enabled:\&{ds.userRoot.enabled\|true}
set-replication-domain-prop --provider-name Multimaster\ Synchronization \
 --domain-name cn=admin\ data --set replication-server:\&{ds.replication.servers}
set-replication-domain-prop --provider-name Multimaster\ Synchronization \
 --domain-name cn=schema --set replication-server:\&{ds.replication.servers} \
 --set enabled:\&{ds.schema.replication.enabled\|false}
set-replication-domain-prop --provider-name Multimaster\ Synchronization \
 --domain-name dc=example,dc=com --set replication-server:\&{ds.replication.servers}
set-replication-server-prop --provider-name Multimaster\ Synchronization \
 --set replication-server:\&{ds.replication.servers}
EOF
```

Templating a Server Configuration With Expressions

Configuration Property	Notes	Example Command

`server-id`	Set the global server ID to an expression such as `&{ds.server.id}`.
	
	The expression `ds.server.id` is set to the actual server ID in the environment. For example, `DS_SERVER_ID=1`.
		`$ dsconfig \`
		` set-global-configuration-prop \`
		` --set server-id:\&{ds.server.id} \`
		` --offline \`
		` --configFile /path/to/opendj/config/config.ldif \`
		` --no-prompt`

`server-fqdn`	Set the server FQDN to an expression such as `&{ds.server.fqdn}`.
	
	This value is used for DIGEST-MD5 SASL authentication. Unless your deployment requires DIGEST-MD5 SASL authentication, disable this mechanism as recommended (`--set enabled: false`).
	
	The expression `ds.server.fqdn` is set to the actual FQDN in the environment. For example, `DS_SERVER_FQDN=ds-rs-1.example.com`.
		`$ dsconfig \`
		` set-sasl-mechanism-handler-prop \`
		` --handler-name DIGEST-MD5 \`
		` --set server-fqdn:\&{ds.fqdn} \`
		` --offline \`
		` --configFile /path/to/opendj/config/config.ldif \`
		` --no-prompt`

Configuration Property	Notes	Example Command

`enabled`	*For each backend holding application data*, set the value to an expression such as `&{ds.backend-name.enabled|true}`.
	
	The expression `ds.backend-name.enabled` is set to a boolean value. For example, `DS_USERROOT_ENABLED=true`.
		`$ dsconfig \`
		` set-backend-prop \`
		` --backend-name userRoot \`
		` --set enabled:\&{ds.userRoot.enabled\\|`
		`true} \`
		` --offline \`
		` --configFile /path/to/opendj/config/`
		`config.ldif \`
		` --no-prompt`

	For each replication domain involving application data, set the value to an expression such as `&{ds.backend-name.enabled|true}`.
	
	*For the `cn=schema` domain*, set the value to an expression such as `&{ds.schema.replication.enabled|false}`.
		`$ dsconfig \`
		` set-replication-domain-prop \`
		` --provider-name Multimaster\`
		` Synchronization \`
		` --domain-name dc=example,dc=com \`
		` --set enabled:\&{ds.userRoot.enabled\\|`
		`true} \`
		` --offline \`
		` --configFile /path/to/opendj/config/`
		`config.ldif \`
		` --no-prompt`

Configuration Property	**Notes**	**Example Command**

`replication-server`	*For each replication domain (including the `cn=admin data` and `cn=schema` domains)*, set the value to an expression such as `&{ds.replication.servers}`.
	
	The expression `ds.replication.servers` is set to an array of *`fqdn:port`* values. For example, `DS_REPLICATION_SERVERS=ds-rs-1.example.com:8989,ds-rs-2.example.com:8989`.
		`$ dsconfig \`
		` set-replication-domain-prop \`
		` --provider-name Multimaster\`
		` Synchronization \`
		` --domain-name cn=admin\ data \`
		` --set replication-server:`
		`\&{ds.replication.servers} \`
		` --offline \`
		` --configFile /path/to/opendj/config/`
		`config.ldif \`
		` --no-prompt`
		`$ dsconfig \`
		` set-replication-domain-prop \`
		` --provider-name Multimaster\`
		` Synchronization \`
		` --domain-name cn=schema \`
		` --set enabled:`
		`\&{ds.schema.replication.enabled\|false} \`
		` --set replication-server:`
		`\&{ds.replication.servers} \`
		` --offline \`
		` --configFile /path/to/opendj/config/`
		`config.ldif \`
		` --no-prompt`
		`$ dsconfig \`
		` set-replication-domain-prop \`
		` --provider-name Multimaster\`
		` Synchronization \`
		` --domain-name dc=example,dc=com \`
		` --set replication-server:`
		`\&{ds.replication.servers} \`
		` --offline \`
		` --configFile /path/to/opendj/config/`
		`config.ldif \`
		` --no-prompt`

	For the local replication server, set the value to an expression such as `&{ds.replication.servers}`.	`$ dsconfig \`
		` set-replication-server-prop \`
		` --provider-name Multimaster\`
		` Synchronization \`
		` --set replication-server:`
		`\&{ds.replication.servers} \`
		` --offline \`
		` --configFile /path/to/opendj/config/`
		`config.ldif \`
		` --no-prompt`

At this point, your server is ready to be used as a template. For additional details on setting expression values before starting the server, see "Using Configuration Property Value Substitution" in the *Administration Guide*.

Applying Further Configuration Changes

The recommended way to apply further configuration changes in your DevOps development environment is to run the **dsconfig** command interactively, and store the changed configuration file `config/config.ldif` under source control.

When run in interactive mode, the **dsconfig** command displays the equivalent command for each update you make. You can store the commands displayed in your source control commit messages that explain each change.

Editing the configuration manually is not recommended, as `config/config.ldif` is not a supported public interface. It can change without notice between releases.

Deploying in a Test or Production Environment

The following sections provide tips for procedures in a test or production environment.

Initializing Data for the First Time

The very first time that you build a new service in an environment, you might already have existing application data. If you do not, you can skip this section.

If you do have data, transform it to LDIF, and do the following to initialize the service with your application data:

1. Deploy the DS servers, but do not start them.
2. For each backend on each server, mount an (empty) backend directory from the persistent volume where directory data will be stored.
3. Import exactly the same LDIF offline for each application data backend on each replica.
4. Once the import has completed successfully for every backend on each replica, start each server.

Backing Up and Restoring Data

Backup and restore operations are the same in DevOps deployments as in static deployments. DevOps deployments will automate the restore process as much as possible to limit administrator interaction.

When you bring up a new replica in an existing environment to replace an existing replica, mount the replica's data from persistent storage at the proper mount points and start the server. Only initialize from backup if you suspect the persisted data is corrupt.

If the new replica is an upgrade, perform a data-only upgrade before starting the server as described in "Upgrading".

Adding a Replica

When adding a replica based on your template, you add a server that is both a directory server and a replication server. You must initialize the data and update the runtime configuration of other replicas to make them aware of the new replication server.

Perform the following high-level steps:

1. Add the server with the proper environment, initializing from backup as described in "To Initialize a Replica From Backup Files" in the *Administration Guide*.

2. For best results, restart each other replica with the same environment as the new replica.

   For example, if you set the expression value `ds.replication.servers` using the environment variable `DS_REPLICATION_SERVERS`, add the FQDN of the new server to the list of replication servers when restarting each server.

Adding a Base DN

When adding a new base DN for new application data, you perform the configuration update separately from the data update.

- Prepare the configuration in your development environment.

- If the new base DN does not yet contain data, create the base entry in the dev environment, and copy the backend database to persistent storage in other environments before restarting the servers. Your startup script should then mount the new backend from persistent storage alongside the other backends.

- If the new base DN holds existing data, initialize the data on a scratch server deployed in the proper environment, and run a full backup.

  After you deploy the build with the configuration for the new base DN, restore from backup on each replica.

Monitoring

As described in "*Monitoring, Logging, and Alerts*" in the *Administration Guide*, DS servers expose monitoring data over HTTP, LDAP, JMX, and SNMP, and can send alerts over JMX or SMTP (email).

HTTP monitoring supports Prometheus and Graphite, as well as custom solutions.

Upgrading

In this context, *upgrade* means unpacking new DS software over the existing, installed files, and then running the **upgrade** command. If you have custom extensions such as server plugins, upgrade them during this phase.

The **upgrade** command adapts the server configuration and other data for use with the new distribution. The process can be quick when only the configuration data has changed in the new release, or relatively slow when the underlying database implementation has changed formats and the **upgrade** command must rebuild all indexes for a large directory data set.

In a DevOps deployment, you maintain the data in different environments separately from the server software with its immutable configuration. This means even if you can upgrade everything at once in your development environment, you must upgrade the data sequentially in your test and production environments. After you mount pre-upgrade data with an upgraded server, and before starting the server, run the **upgrade** command with the `--dataOnly` option:

```
$ upgrade --dataOnly
```

You upgrade replicas with a rolling upgrade process as described in "To Upgrade Replicated Servers" in the *Installation Guide*.

Chapter 7 Deployment Checklists

This chapter offers checklists for deploying a directory service.

Initiating the Project

Task	Done?
Understand the business requirements for your DS deployment	☐
Identify key client applications	☐
Identify project stakeholders	☐
Define SLOs based on business requirements	☐
Define project scope	☐
Define project roles and responsibilities	☐
Schedule DS training for deployment team members	☐

Preparing Supportability

Task	Done?
Find out how to get help and support from ForgeRock and partners	☐
Find out how to get training from ForgeRock and partners	☐
Find out how to keep up to date with new development and new releases	☐
Find out how to report problems	☐

Designing the Directory Service

Task	Done?
Understand the roles of directory components	☐
Define architecture, mapping requirements to component features	☐
Define the directory data model	☐
Define the directory access model	☐
Define the replication model	☐
Define how to backup, restore, and recover data	☐
Define how you will monitor and audit the service	☐

Task	Done?
Determine how to harden and secure the service	☐

Developing the Directory Service

Task	Done?
Engage development of custom server plugins as necessary	☐
Apply configuration management	☐
Create a test plan	☐
Engage automation, continuous integration	☐
Create a documentation plan	☐
Create a maintenance and support plan	☐
Pilot the implementation	☐
Size systems to provision for production	☐
Execute test plans	☐
Execute documentation plans	☐
Create a rollout plan in alignment with all stakeholders	☐
Prepare patch and upgrade plans	☐

Implementing the Directory Service

Task	Done?
Ensure appropriate support for production services	☐
Execute the rollout plan	☐
Engage ongoing monitoring and auditing services	☐
Engage ongoing maintenance and support	☐

Maintaining the Directory Service

Task	Done?
Execute patch and upgrade plans as necessary	☐
Plan how to adapt the deployment to new and changing requirements	☐

Appendix A. Getting Support

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details on ForgeRock's support offering, including support plans and service level agreements (SLAs), visit https://www.forgerock.com/support.

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Glossary

Abandon operation	LDAP operation to stop processing of a request in progress, after which the server drops the connection without a reply to the client application.
Access control	Control to grant or to deny access to a resource.
Access control instruction (ACI)	Instruction added as a directory entry attribute for fine-grained control over what a given user or group member is authorized to do in terms of LDAP operations and access to user data. ACIs are implemented independently from privileges, which apply to administrative operations. See also Privilege.
Access control list (ACL)	An access control list connects a user or group of users to one or more security entitlements. For example, users in group sales are granted the entitlement read-only to some financial data.
`access` log	Server log tracing the operations the server processes including timestamps, connection information, and information about the operation itself.
Account lockout	The act of making an account temporarily or permanently inactive after successive authentication failures.
Active user	A user that has the ability to authenticate and use the services, having valid credentials.
Add operation	LDAP operation to add a new entry or entries to the directory.

Anonymous	A user that does not need to authenticate, and is unknown to the system.
Anonymous bind	A bind operation using simple authentication with an empty DN and an empty password, allowing anonymous access such as reading public information.
Approximate index	Index is used to match values that "sound like" those provided in the filter.
Attribute	Properties of a directory entry, stored as one or more key-value pairs. Typical examples include the common name (`cn`) to store the user's full name and variations of the name, user ID (`uid`) to store a unique identifier for the entry, and `mail` to store email addresses.
`audit` log	Type of access log that dumps changes in LDIF.
Authentication	The process of verifying who is requesting access to a resource; the act of confirming the identity of a principal.
Authorization	The process of determining whether access should be granted to an individual based on information about that individual; the act of determining whether to grant or to deny a principal access to a resource.
Backend	Repository that stores directory data. Different implementations with different capabilities exist.
Binary copy	Binary backup archive of one directory server that can be restored on another directory server.
Bind operation	LDAP authentication operation to determine the client's identity in LDAP terms, the identity which is later used by the server to authorize (or not) access to directory data that the client wants to lookup or change.
Branch	The distinguished name (DN) of a non-leaf entry in the Directory Information Tree (DIT), and also that entry and all its subordinates taken together. Some administrative operations allow you to include or exclude branches by specifying the DN of the branch. See also Suffix.
Collective attribute	A standard mechanism for defining attributes that appear on all the entries in a particular subtree.
Compare operation	LDAP operation to compare a specified attribute value with the value stored on an entry in the directory.

Control	Information added to an LDAP message to further specify how an LDAP operation should be processed. DS supports many LDAP controls.
Database cache	Memory space set aside to hold database content.
`debug` log	Server log tracing details needed to troubleshoot a problem in the server.
Delete operation	LDAP operation to remove an existing entry or entries from the directory.
Directory	A directory is a network service which lists participants in the network such as users, computers, printers, and groups. The directory provides a convenient, centralized, and robust mechanism for publishing and consuming information about network participants.
Directory hierarchy	A directory can be organized into a hierarchy in order to make it easier to browse or manage. Directory hierarchies normally represent something in the physical world, such as organizational hierarchies or physical locations. For example, the top level of a directory may represent a company, the next level down divisions, the next level down departments, and down the hierarchy. Alternately, the top level may represent the world, the next level down countries, next states or provinces, and next cities.
Directory Information Tree (DIT)	A set of directory entries organized hierarchically in a tree structure, where the vertices are the entries and the arcs between vertices define relationships between entries
Directory manager	Default directory superuser who has privileges to do full administration of the DS server, including bypassing access control evaluation, changing access controls, and changing administrative privileges. See also Superuser.
Directory object	A directory object is an item in a directory. Example objects include users, user groups, computers, and more. Objects may be organized into a hierarchy and contain identifying attributes. See also Entry.
Directory proxy server	Server that forwards LDAP requests to remote directory servers. A standalone directory proxy server does not store user data. See also Directory server.
Directory server	Server application for centralizing information about network participants. A highly available directory service consists of multiple directory servers configured to replicate directory data. See also Directory, Replication.

Directory Services Markup Language (DSML)	Standard language to access directory services using XML. DMSL v1 defined an XML mapping of LDAP objects, while DSMLv2 maps the LDAP Protocol and data model to XML.
Distinguished name (DN)	Fully qualified name for a directory entry, such as `uid=bjensen, ou=People,dc=example,dc=com`, built by concatenating the entry RDN (`uid=bjensen`) with the DN of the parent entry (`ou=People,dc=example, dc=com`).
Domain	A replication domain consists of several directory servers sharing the same synchronized set of data. The base DN of a replication domain specifies the base DN of the replicated data.
DSML gateway	Standalone web application that translates DSML requests from client applications to LDAP requests to a directory service, and LDAP responses from a directory service to DSML responses to client applications.
Dynamic group	Group that specifies members using LDAP URLs.
Entry	As generic and hierarchical data stores, directories always contain different kinds of entries, either nodes (or containers) or leaf entries. An entry is an object in the directory, defined by one of more object classes and their related attributes. At startup, DS servers report the number of entries contained in each suffix.
Entry cache	Memory space set aside to hold frequently accessed, large entries, such as static groups.
Equality index	Index used to match values that correspond exactly (though generally without case sensitivity) to the value provided in the search filter.
`errors` log	Server log tracing server events, error conditions, and warnings, categorized and identified by severity.
Export	Save directory data in an LDIF file.
Extended operation	Additional LDAP operation not included in the original standards. DS servers support several standard LDAP extended operations.
Extensible match index	Index for a matching rule other than approximate, equality, ordering, presence, substring or VLV, such as an index for generalized time.
External user	An individual that accesses company resources or services but is not working for the company. Typically a customer or partner.
Etime	Elapsed time within the server to process a request, starting from the moment the decoded operation is available to be processed by a worker thread.

Filter	An LDAP search filter is an expression that the server uses to find entries that match a search request, such as `(mail=*@example.com)` to match all entries having an email address in the example.com domain.
Group	Entry identifying a set of members whose entries are also in the directory.
Idle time limit	Defines how long DS allows idle connections to remain open.
Import	Read in and index directory data from an LDIF file.
Inactive user	An entry in the directory that once represented a user but which is now no longer able to be authenticated.
Index	Directory server backend feature to allow quick lookup of entries based on their attribute values. See also Approximate index, Equality index, Extensible match index, Ordering index, Presence index, Substring index, Virtual list view (VLV) index, Index entry limit.
Index entry limit	When the number of entries that an index key points to exceeds the index entry limit, DS stops maintaining the list of entries for that index key.
Internal user	An individual who works within the company either as an employee or as a contractor.
LDAP Data Interchange Format (LDIF)	Standard, portable, text-based representation of directory content. See RFC 2849.
LDAP URL	LDAP Uniform Resource Locator such as `ldap://directory.example.com:389/dc=example,dc=com??sub?(uid=bjensen)`. See RFC 2255.
LDAPS	LDAP over SSL.
Lightweight Directory Access Protocol (LDAP)	A simple and standardized network protocol used by applications to connect to a directory, search for objects and add, edit or remove objects. See RFC 4510.
Lookthrough limit	Defines the maximum number of candidate entries DS considers when processing a search.
Matching rule	Defines rules for performing matching operations against assertion values. Matching rules are frequently associated with an attribute syntax and are used to compare values according to that syntax. For example, the `distinguishedNameEqualityMatch` matching rule can be used to determine whether two DNs are equal and can ignore unnecessary spaces around commas and equal signs, differences in capitalization in attribute names, and other discrepancies.

Modify DN operation	LDAP modification operation to request that the server change the distinguished name of an entry.
Modify operation	LDAP modification operation to request that the server change one or more attributes of an entry.
Naming context	Base DN under which client applications can look for user data.
Object class	Identifies entries that share certain characteristics. Most commonly, an entry's object classes define the attributes that must and may be present on the entry. Object classes are stored on entries as values of the `objectClass` attribute. Object classes are defined in the directory schema, and can be abstract (defining characteristics for other object classes to inherit), structural (defining the basic structure of an entry, one structural inheritance per entry), or auxiliary (for decorating entries already having a structural object class with other required and optional attributes).
Object identifier (OID)	String that uniquely identifies an object, such as `0.9.2342.19200300.100.1.1` for the user ID attribute or `1.3.6.1.4.1.1466.115.121.1.15` for `DirectoryString` syntax.
Operational attribute	An attribute that has a special (operational) meaning for the server, such as `pwdPolicySubentry` or `modifyTimestamp`.
Ordering index	Index used to match values for a filter that specifies a range.
Password policy	A set of rules regarding what sequence of characters constitutes an acceptable password. Acceptable passwords are generally those that would be too difficult for another user or an automated program to guess and thereby defeat the password mechanism. Password policies may require a minimum length, a mixture of different types of characters (lowercase, uppercase, digits, punctuation marks, and other characters), avoiding dictionary words or passwords based on the user's name, and other attributes. Password policies may also require that users not reuse old passwords and that users change their passwords regularly.
Password reset	Password change performed by a user other than the user who owns the entry.
Password storage scheme	Mechanism for encoding user passwords stored on directory entries. DS implements a number of password storage schemes.
Password validator	Mechanism for determining whether a proposed password is acceptable for use. DS implements a number of password validators.
Plugin	Java library with accompanying configuration that implements a feature through processing that is not essential to the core operation of DS servers.

	As the name indicates, plugins can be plugged in to an installed server for immediate configuration and use without recompiling the server. DS servers invoke plugins at specific points in the lifecycle of a client request. The DS configuration framework lets directory administrators manage plugins with the same tools used to manage the server.
Presence index	Index used to match the fact that an attribute is present on the entry, regardless of the value.
Principal	Entity that can be authenticated, such as a user, a device, or an application.
Privilege	Server configuration settings controlling access to administrative operations such as exporting and importing data, restarting the server, performing password reset, and changing the server configuration. Privileges are implemented independently from access control instructions (ACI), which apply to LDAP operations and user data. See also Access control instruction (ACI).
Referential integrity	Ensuring that group membership remains consistent following changes to member entries.
`referint` log	Server log tracing referential integrity events, with entries similar to the errors log.
Referral	Reference to another directory location, which can be another directory server running elsewhere or another container on the same server, where the current operation can be processed.
Relative distinguished name (RDN)	Initial portion of a DN that distinguishes the entry from all other entries at the same level, such as `uid=bjensen` in `uid=bjensen,ou=People,dc=example,dc=com`.
Replica	Directory server this is configured to use replication.
Replication	Data synchronization that ensures all directory servers participating eventually share a consistent set of directory data.
`replication` log	Server log tracing replication events, with entries similar to the errors log.
Replication server	Server dedicated to transmitting replication messages. A standalone replication server does not store user data.
REST to LDAP gateway	Standalone web application that translates RESTful HTTP requests from client applications to LDAP requests to directory services, and

	LDAP responses from directory services to HTTP responses to client applications.
Root DSE	The directory entry with distinguished name "" (empty string), where DSE is an acronym for DSA-Specific Entry. DSA is an acronym for Directory Server Agent, a single directory server. The root DSE serves to expose information over LDAP about what the directory server supports in terms of LDAP controls, auth password schemes, SASL mechanisms, LDAP protocol versions, naming contexts, features, LDAP extended operations, and other information.
Schema	LDAP schema defines the object classes, attributes types, attribute value syntaxes, matching rules and other constrains on entries held by the directory server.
Search filter	See Filter.
Search operation	LDAP lookup operation where a client requests that the server return entries based on an LDAP filter and a base DN under which to search.
Simple authentication	Bind operation performed with a user's entry DN and user's password. Use simple authentication only if the network connection is secure.
Size limit	Sets the maximum number of entries returned for a search.
Static group	Group that enumerates member entries.
Subentry	An entry, such as a password policy entry, that resides with the user data but holds operational data, and is not visible in search results unless explicitly requested.
Substring index	Index used to match values specified with wildcards in the filter.
Suffix	The distinguished name (DN) of a root entry in the Directory Information Tree (DIT), and also that entry and all its subordinates taken together as a single object of administrative tasks such as export, import, indexing, and replication.
Superuser	User with privileges to perform unconstrained administrative actions on DS server. This account is analogous to the UNIX `root` and Windows `Administrator` accounts. Superuser privileges include the following: • `bypass-acl`: The holder is not subject to access control. • `privilege-change`: The holder can edit administrative privileges. • `proxied-auth`: The holder can make requests on behalf of another user, including directory superusers.

	The default superuser is `cn=Directory Manager`. You can create additional superuser accounts, each with different administrative privileges. See also Directory manager, Privilege.
Task	Mechanism to provide remote access to server administrative functions. DS software supports tasks to back up and restore backends, to import and export LDIF files, and to stop and restart the server.
Time limit	Defines the maximum processing time DS devotes to a search operation.
Unbind operation	LDAP operation to release resources at the end of a session.
Unindexed search	Search operation for which no matching index is available. If no indexes are applicable, then the directory server potentially has to go through all entries to look for candidate matches. For this reason, the `unindexed-search` privilege, which allows users to request searches for which no applicable index exists, is reserved for the directory manager by default.
User	An entry that represents an individual that can be authenticated through credentials contained or referenced by its attributes. A user may represent an internal user or an external user, and may be an active user or an inactive user.
User attribute	An attribute for storing user data on a directory entry such as `mail` or `givenname`.
Virtual attribute	An attribute with dynamically generated values that appear in entries but are not persistently stored in the backend.
Virtual directory	An application that exposes a consolidated view of multiple physical directories over an LDAP interface. Consumers of the directory information connect to the virtual directory's LDAP service. Behind the scenes, requests for information and updates to the directory are sent to one or more physical directories where the actual information resides. Virtual directories enable organizations to create a consolidated view of information that for legal or technical reasons cannot be consolidated into a single physical copy.
Virtual list view (VLV) index	Browsing index designed to help the directory server respond to client applications that need, for example, to browse through a long list of results a page at a time in a GUI.
Virtual static group	DS group that lets applications see dynamic groups as what appear to be static groups.

X.500	A family of standardized protocols for accessing, browsing and maintaining a directory. X.500 is functionally similar to LDAP, but is generally considered to be more complex, and has consequently not been widely adopted.