

Deployment

This guide focuses on how to use Directory Services software to build secure, high-performance, manageable directory services. It helps directory service architects design scalable services that fit their needs.



DS Software

Use DS components.



Project Outline

Outline a successful plan.



Complete Plans

Create comprehensive plans.



Deployment Patterns

Apply best practices.




Provisioning

Prepare systems and hardware.



Checklists

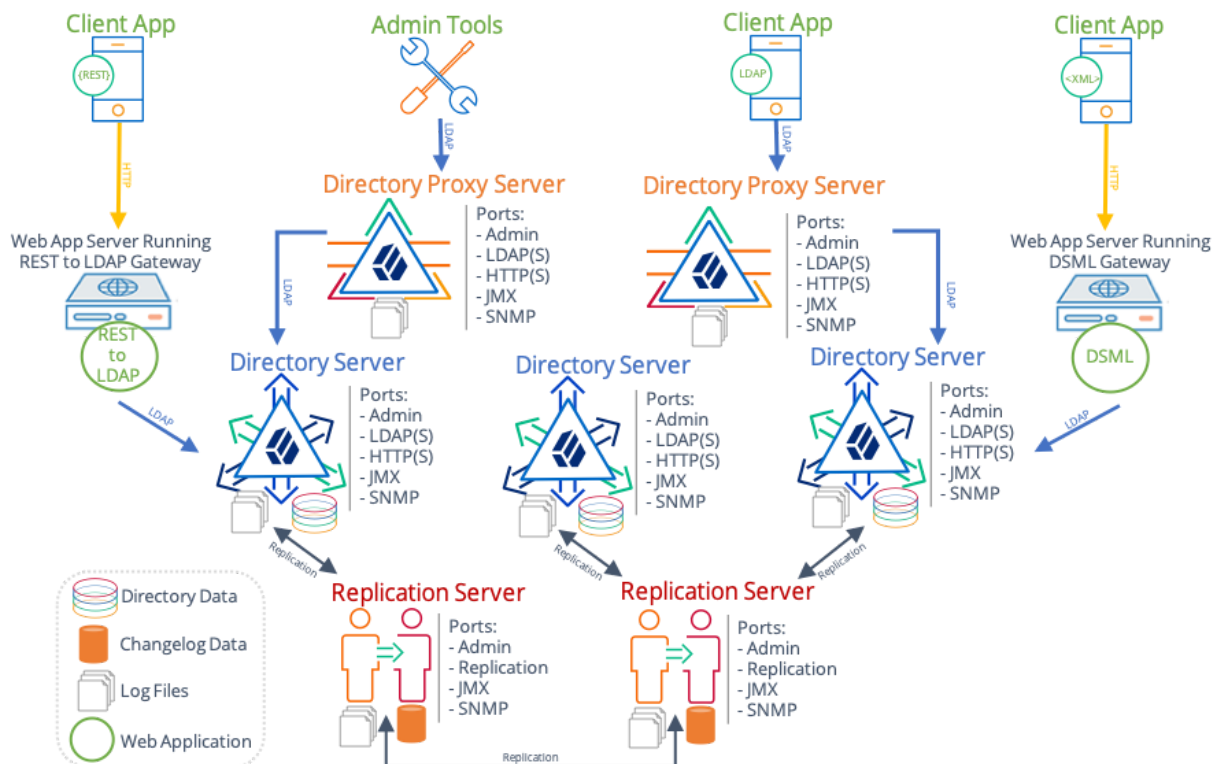
Follow checklists.

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

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

DS Software

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:



- *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 Servers

Directory servers have the following characteristics.

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.

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 [Data Storage](#).

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](#).

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 [Logging](#). As a result of default retention policies, messages can eventually be lost unless you copy old files to another system for permanent storage.

Backup files

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

For details, see [Import and Export](#), and [Backup and Restore](#).

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 requires more system resources than a small database serving one, simple 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 or LDIF 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

Standalone replication servers have the following characteristics. 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.

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.

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 [Changelog for Notifications](#).

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 [Logging](#). This means, however, that messages are eventually lost unless you move old files to another system for permanent storage.

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

Directory proxy servers have the following characteristics.

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](#).

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 [Logging](#). This means, however, that messages are eventually lost unless you move old files to another system for permanent storage.

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.

Command-Line 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 Commands](#).

DSML Gateway

The standalone DSML gateway web application has the following characteristics.

You can install this component independently of directory services. For details, see [Install a DSML Gateway](#).

Roles

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 container. It is limited to one host:port combination for the LDAP directory service.

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 container for other services, such as logging.

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

The standalone REST to LDAP gateway web application has the following characteristics. 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 [Install a 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 container. 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.

Data

A REST to LDAP gateway maintains only its own service configuration, recorded in files as described in [REST to LDAP Reference](#). It depends on the host web application container for other services, such as logging.

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.

Project Outline

Consider the following when preparing the high-level project plan.

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.

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.

Important Questions

- **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 [Security](#).

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 with the evaluation setup profile, be aware that access control settings for Example.com data in the evaluation setup profile are very lenient.

- **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.

Planning Steps

Follow these steps to a successful deployment.

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 all the 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 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 test 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.

Comprehensive Plans

Your comprehensive deployment plan should cover the following themes.

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.

Customization

DS servers provide a Java plugin API that allows you to extend and customize server processing. A server plugin is a library that you plug into an installed server and configure for use. The DS server calls the plugin as described in Plugin Types.

DS servers have many features that are implemented as server plugin extensions. 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 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.

NOTE

ForgeRock supports customers using standard plugins delivered as part of DS software.

If you deploy with custom plugins and need support in production, contact info@forgerock.com in advance to determine how your deployment can be supported.

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. For details, see Tests.

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

Plugin Types

Plugin types correspond to the points where the server invokes the plugin.

For the full list of plugin invocation points, see the Javadoc for [PluginType](#). The following list summarizes the plugin invocation points:

- At server startup and shutdown
- Before and after data export and import
- Immediately after a client connection is established or is closed
- Before processing begins on an LDAP operation (to change an incoming request before it is decoded)
- Before core processing for LDAP operations (to change the way the server handles the operation)
- After core processing for LDAP operations (where the plugin can access all information about the operation including the impact it has on the targeted entry)
- When a subordinate entry is deleted as part of a subtree delete, or moved or renamed as part of a modify DN operation
- Before sending intermediate and search responses
- After sending a result

A plugin's types are specified in its configuration, and can therefore be modified at runtime.

Plugin Configuration

Server plugin configuration is managed with the same configuration framework that is used for DS server configuration.

The DS configuration framework has these characteristics:

- LDAP schemas govern what attributes can be used in plugin configuration entries.

For all configuration attributes that are specific to a plugin, the plugin should have its own object class and attributes defined in the server LDAP schema. Having configuration entries governed by schemas makes it possible for the server to identify and prevent configuration errors.

For plugins, having schema for configuration attributes means that an important part of plugin installation is making the schema definitions available to the DS server.

- The plugin configuration is declared in XML files.

The XML specifies configuration properties and their documentation, and inheritance relationships.

The XML Schema Definition files (.xsd files) for the namespaces used are not published externally. For example, the namespace identifier

`http://opendj.forgerock.org/admin` is not an active URL. An XML configuration definition has these characteristics:

- The attributes of the `<managed-object>` element define XML namespaces, a (singular) name and plural name for the plugin, and the Java-related inheritance of the implementation to generate. A *managed object* is a configurable component of DS servers.

A managed object definition covers the object's structure and inheritance, and is like a class in Java. The actual managed object is like an instance of an object in Java. Its configuration maps to a single LDAP entry in the configuration backend `cn=config`.

Notice that the `<profile>` element defines how the whole object maps to an LDAP entry in the configuration. The `<profile>` element is mandatory, and should include an LDAP profile.

The `name` and `plural-name` properties are used to identify the managed object definition. They are also used when generating Java class names. Names must be a lowercase sequence of words separated by hyphens.

The `package` property specifies the Java package name for generated code.

The `extends` property identifies a parent definition that the current definition inherits.

- The mandatory `<synopsis>` element provides a brief description of the managed object.

If a longer description is required, add a `<description>`. The `<description>` is used in addition to the synopsis, so there is no need to duplicate the synopsis in the description.

- The `<property>` element defines a property specific to the plugin, including its purpose, its default value, its type, and how the property maps to an LDAP attribute in the configuration entry.

The `name` attribute is used to identify the property in the configuration.

- The `<property-override>` element sets the pre-defined property `java-class` to the fully qualified implementation class.
- Compilation generates the server-side and client-side APIs to access the plugin configuration from the XML. To use the server-side APIs in a plugin project, first generate and compile them, and then include the classes on the project classpath.

When a plugin is loaded in the DS server, the client-side APIs are available to configuration tools like the **dsconfig** command. Directory administrators can configure a custom plugin in the same way they configure other server components.

- The framework supports internationalization.

The plugin implementation selects appropriate messages from the resource bundle based on the server locale. If no message is available for the server locale, the plugin falls back to the default locale.

A complete plugin project includes LDAP schema definitions, XML configuration definitions, Java plugin code, and Java resource bundles. For examples, see the sample plugins delivered with DS software.

Pilot Projects

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 investing too much, so 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 experiment with 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.

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. You must have or be able to generate representative data samples for planning purposes. You must be able to 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 LDAP Schema Extensions.

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

See The DIT.

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

See 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 Data Management.

- What groups will be defined in your directory service?

See Groups.

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

See Shared Data.

- How should you cache data for appropriate performance?

See Caching.

- How will identities be managed in your deployment?

See Identity Management.

LDAP Schema Extensions

As described in [LDAP Schema](#), 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 [Update LDAP Schema](#).

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 for updates, 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 [Schema Reference](#).

4. Define new LDAP schema definitions for data that does not fit existing definitions. This is described in [About LDAP Schema](#), and [Update LDAP Schema](#).

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.

The DIT

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

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

For example, you can build base DN 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 DN 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 DN 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 hierarchical branches 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 hierarchical branches 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, ACLs 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](#), the primary unit of replication is the base DN. If necessary, you can split a base DN into multiple branches. For example use cases, read [Deployment Patterns](#).

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

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 with client hostnames or address masks. 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.

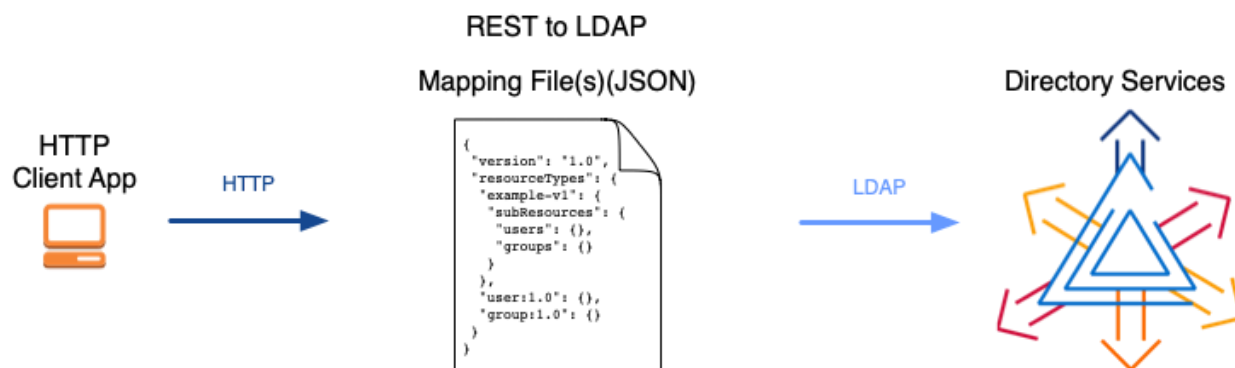


Figure 1. Mapping JSON Resources to LDAP Entries

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 Reference](#).

Data Management

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 data management is, however, often a task shared by multiple users. 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. 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 [Administrative Roles](#) to review the alternatives.

Groups

As described in [Groups](#), DS directory servers offer dynamic, static, and virtual static group implementations:

- Dynamic groups identify members with an LDAP URL.

An entry belongs to a dynamic group when it matches the base DN, scope, and filter defined in a member URL of the group. Changes to the entry can modify its dynamic group membership.

- Static groups enumerate each member. The size of a static group entry can grow very large in a high-scale directory.

- 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 [Cache for Large Groups](#).

Shared Data

As described in [Virtual Attributes](#), and [Collective Attributes](#), 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

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

A disadvantage of caching all data is that systems with enough RAM are more expensive. Consider the suggestions in [Database Cache Settings](#), testing the results for your data when planning your deployment.

Identity Management

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	<ul style="list-style-type: none"> • Accounts • Active Accounts
Authentication	<ul style="list-style-type: none"> • Authentication Mechanisms • Authentication (Binds) • Certificate-Based Authentication • Pass-Through Authentication • DS REST APIs
Authorization	<ul style="list-style-type: none"> • Configure HTTP Authorization • Proxied Authorization

Topics	References
Password Management	<ul style="list-style-type: none"> • Password management • Changing passwords over LDAP • Changing passwords over HTTP

Directory Access

Consider these topics when designing the access model for your deployment.

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 [Administrative Access](#), 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 [The DIT](#).

Immutable and 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 [Manual Initialization](#), new replicas can start with existing data sets.

Fine-Grained Access

DS servers provide both HTTP and LDAP access to directory data. 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, and through access controls. For details, see [Access Control](#).

Access control instructions (ACIs) in directory data take the form of `aci` LDAP attributes, or `global-aci` properties in the server configuration. You write ACIs in a domain-specific language. 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.

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 details, see [Administrative Privileges](#).

Take care when granting privileges, especially the following privileges:

- `bypass-ac1`: The holder is not subject to access control.
- `config-write`: The holder can edit the server configuration.
- `modify-ac1`: 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 `uid=admin`.

Authentication

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 details, see [Authentication Mechanisms](#).

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 [Access Control](#).

As mentioned in [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 [LDAP Proxy](#).

HTTP Access

As described in [HTTP Access](#), 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 [Data Views](#). For details, see [REST to LDAP 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](#).

Data Replication

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

Consistency and 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, but clients making requests 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 [Load Balancing](#).

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 [Install Standalone Servers](#) and [Replication Groups](#).

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 [Manual Initialization](#). Once a replica has been initialized, replication eventually brings its data into a consistent state with the other replicas. As described in [Consistency and 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](#), [Optimistic Concurrency \(MVCC\)](#), and [Update](#). To work around broken client applications that assume immediate consistency, use affinity load balancing in directory proxy servers. For details, see [Load Balancing](#).

Some client applications need notifications when directory data changes. Client applications cannot participate in replication itself, but can get change notifications. For details, see [Changelog for Notifications](#).

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).

High 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 [High Availability](#).

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.

Before finalizing deployment plans, make sure that you understand key availability features in detail. For details about replication, read [Replication](#). For details about proxy features, read [LDAP Proxy](#).

Backup and Recovery

Make sure your plans define how you:

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

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 [Data Storage](#).

Backup and restore procedures are described in [Backup and Restore](#). When planning your backup and recovery strategies, be aware of the following key features:

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

- 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 replicated data from backup, the replication protocol brings the replica up to date with others after the restore operation.

This requires, however, that the backup is recent enough. Backup files 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 a serious accidental deletion or change, you must restore the entire directory service to an earlier state.

For details, see [Recover From User Error](#).

- When you restore encrypted data, the server must have the same shared master key as the server that performed the backup.

Otherwise, the directory server cannot decrypt the symmetric key used to decrypt the data. For details, see [Data Encryption](#).

- For portability, you can also export directory data to LDIF files. You can then restore directory data by importing the LDIF.

If you have stored passwords with a reversible encryption password storage scheme, be aware that the server must have the same shared master key as the server that encrypted the password.


- 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

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 [Use Administrative APIs](#).

- LDAP connection handlers expose a `cn=monitor` branch that offers LDAP access to monitoring data.

For details, see [LDAP-Based Monitoring](#).

- JMX connection handlers offer remote access.

For details, see [JMX-Based Monitoring](#).

- SNMP connection handlers enable remote access.

For details, see [SNMP-Based Monitoring](#).

- You can configure alerts to be sent over JMX or SMTP (mail).

For details, see [Alerts](#).

- Replication conflicts are found in the directory data.

For details, see [Replication Conflicts](#).

- Server tools, such as the `status` command, can run remotely.

For details, see [Status and Tasks](#).

The following data sources require access to the server system:

- Server logs, as described in [Logging](#).

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 [Rebuild Indexes](#), and [Verify Indexes](#).

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 files 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 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 remains high and continues to increase over the long term.

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 Security

When you first set up DS servers with the evaluation profile, the configuration favors ease of use over security for Example.com data.

All other configurations and setup profiles leave the server hardened for more security by default. You explicitly grant additional access if necessary.

For additional details, see [Security](#).

Tests

In addition to planning tests for each custom plugin, test each feature you deploy. Perform functional and 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.

Functional Tests

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.

Performance Tests

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 [Performance Tests](#), and [Gatling](#)[☞], which uses a domain specific language for load testing. To mimic the production load, examine the access patterns, and 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 `makeidif` 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.

Deployment Tests

Here, deployment testing is a description rather than a term. 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.

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.

Review the configuration options described here, so that you know what to put 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](#) 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

Method	Notes
Tools (<code>dsconfig</code> 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 <code>var/config.ldif.startok</code> file, and with the compressed snapshots of the main configuration in the <code>var/archived-configs/</code> 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.

Documentation

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 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.

Maintenance and Support

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 Documentation, match their requirements?
- Do they have any additional acceptance criteria for deployment tests, as described in Deployment Tests?

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.

Rollout

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.

Ongoing 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 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 the DIT to replicate different branches separately. Deploy the replicas for each branch on sets of separate systems. For details, see [High Scalability](#).

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. 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 [High Availability](#), assess the risks. In light of the risks, devise and test disaster recovery procedures.

For details, refer to [Disaster recovery](#).

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.

Deployment Patterns

Use these patterns in your deployments.

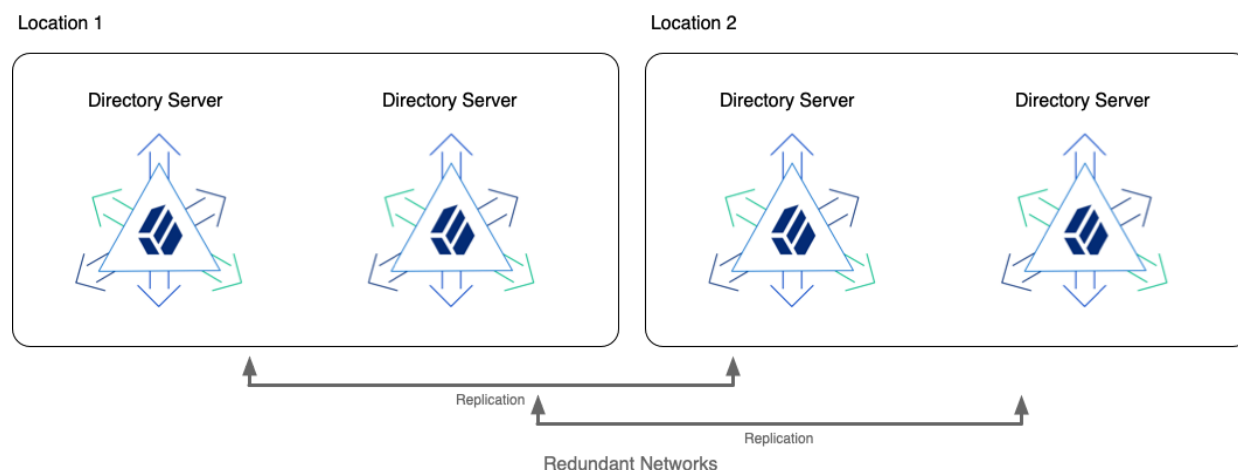
High Availability

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 [Consistency and Availability](#).

- DS upgrade capabilities let you perform rolling upgrades without ever taking the whole service offline.
- If desired, DS proxy capabilities help you provide a single point of entry for directory applications, hiding the fact that individual servers do go offline.



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. Standalone Replication Servers

If you have a large DS service with standalone replication servers, they provide the foundation for high availability. They communicate change messages to directory server replicas, and they also let other servers 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

If you use DS directory proxy servers for a unified view of the service, 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. Make sure you have safe copies of the master keys that let directory servers decrypt encrypted data. For details, see [Backup and Restore](#).

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](#), directory servers might have to replay their transaction logs before they are ready. However, 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. For details, refer to [Disaster recovery](#).

- **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 fail over automatically, and to retry requests for certain types of failure. For details, see [LDAP Proxy](#).

- **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 `isolation-policy` setting 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.

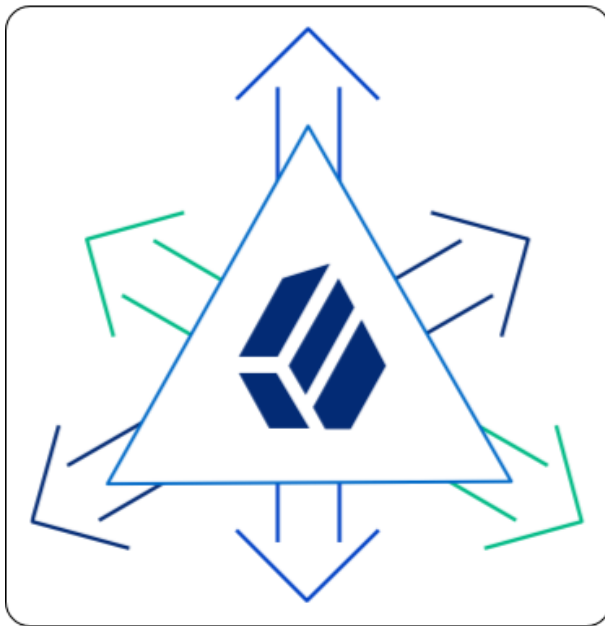
2. Rebuild replication servers.
3. Rebuild directory servers.
4. Rebuild directory proxy servers.

High Scalability

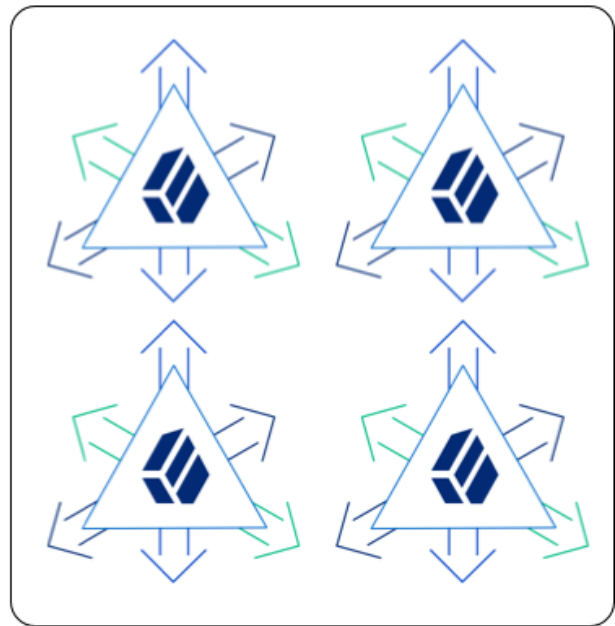
A high-scale directory service is one that requires high performance. For example, very high throughput or very low response times, or both. Or, it has a large data set, such as 100 million entries. When building a high-scale directory, the fundamental question is whether to scale up or scale out.

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

Scale Up



Scale Out



Scale Up or Scale Out

	Scaling Up	Scaling Out
Why Choose...?	<ul style="list-style-type: none">• Simpler architecture• Cannot distribute or shard data	<ul style="list-style-type: none">• Very high update load• Can distribute or shard data
Advantages	<ul style="list-style-type: none">• Simpler architecture• No need to distribute or shard data	<ul style="list-style-type: none">• Not limited by underlying platform• Smaller server systems• Better isolation of issues• High update scalability
Disadvantages	<ul style="list-style-type: none">• Limited by underlying platform• Powerful (expensive) server systems• Less isolation of issues• Limited write scalability	<ul style="list-style-type: none">• Complex architecture• Must distribute/shard data somehow

Plan to Scale

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

Define SLOs as described in [Performance Requirements](#). Once you have defined the SLOs, model directory client traffic to test them using your own tools, or the tools described in [Performance Tests](#).

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 [Generate Test Data](#). 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, make sure the system has enough RAM available to cache the database. By default, database files are stored under the `/path/to/opensj/db` directory. Ideally, the RAM available to the server should be at least 1.5 to 2 times the total size of the database files on disk.

Scale 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 Plan to Scale remain within the capabilities of each server and system.

In addition to the recommendations in [Hardware](#), and the tips in [Performance Settings](#), 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.

Scale 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](#). 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 [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 [Single Point of Access](#).

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:

- 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 [Access Control](#), 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.

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 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.
- For details on replicating only part of the data set outside a given location, see Fractional Replication.

Subtree Replication

As described in [Replication Per Base DN](#), the primary unit of replication is the 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 `userData` backend holds top-level entries, which do not directly reference users in a particular region.
- The `region1` backend holds entries under the `ou=Region 1,dc=example,dc=com` base DN.
- The `region2` backend holds entries under the `ou=Region 2,dc=example,dc=com` base DN.

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 userData --->%
# Base entries are stored in the userData backend:
dn: dc=example,dc=com # Base DN of
userData backend
...

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

# %<--- End of LDIF for userData --->%
# %<--- 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=8Ev1fE0rRa3rgbX0,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=8Ev1fE0rRa3rgbX0,ou=people,ou=Region 2,dc=example,dc=com
uid: 8Ev1fE0rRa3rgbX0
...

# %<--- 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.



Figure 2. 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 base DN in both regions for availability.
- Region 2 regulations do not allow region 2 user data to be replicated to region 1.

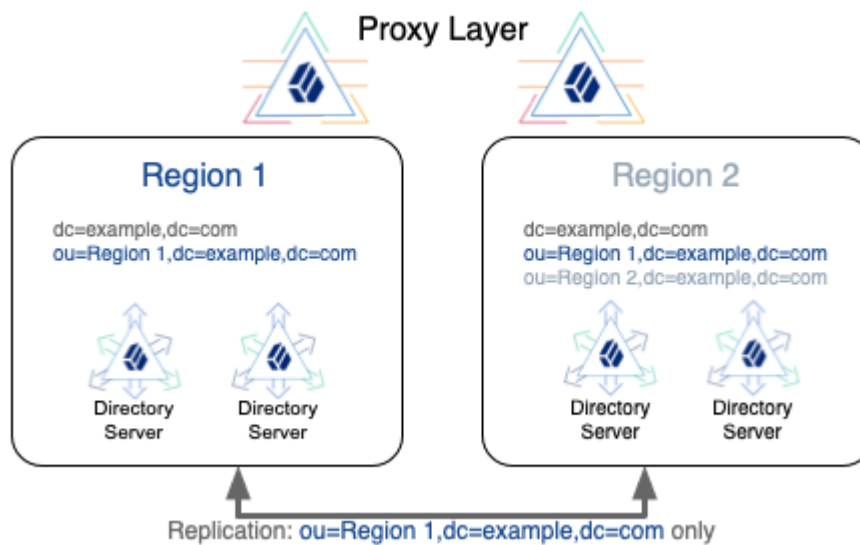


Figure 3. Mixed Replication Domains for Data Sovereignty

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](#), and [Split Data](#).

Fractional Replication

In some deployments, regulations let you replicate some user attributes. For example, consider a deployment where data sovereignty regulations in one region let you replicate UIDs and class of service levels everywhere, but do not let personally identifiable information leave the user's location.

Consider the following entry where you replicate only the `uid` and `classOfService` attributes outside the user's region:

```
dn: uid=aqepfrfEUXIEuMa7M,ou=people,ou=Region 1,dc=example,dc=com
objectClass: top
objectClass: cos
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
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=2jD5Nanz0ZGjMmcz,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
uidNumber: 1076
userpassword: {PBKDF2-HMAC-SHA256}10000:<hash>
# Outside the user's region, you replicate only these attributes:
uid: aqeprefEUXIEuMa7M
classOfService: bronze

```

To let you 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 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.

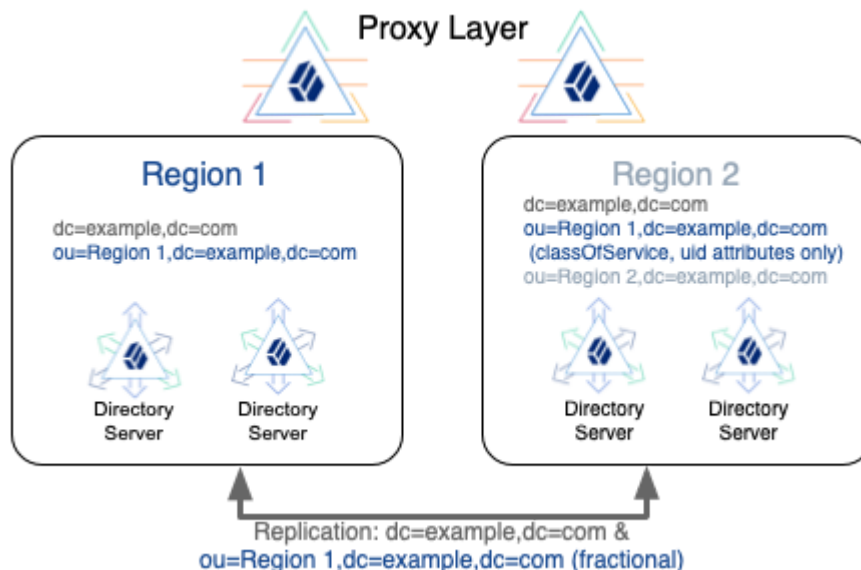


Figure 4. Fractional Replication for Protected Data

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 [LDAP Schema](#).

For additional information, see [Fractional Replication](#).

Interoperability

Common use cases involve interoperability with other directory software.

Use Case	See...
More than one directory service	Proxy Layer
Credentials in another directory service	Pass-Through Authentication
Must sync changes across directory services	Data Synchronization and Migration
Web clients need alternate data views	Alternative Views

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 [Single Point of Access](#).

Pass-Through 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 [Pass-Through Authentication](#).

Data Synchronization and 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.

Alternative Views

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 [Change Incoming Updates](#).)

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 [Configure HTTP User APIs](#), and [REST to LDAP Reference](#).

Provisioning Systems

Before running Directory Services software in production, review the [Requirements](#) page of the *Release Notes*, and the following information.

Sizing Systems

Given availability requirements and estimates on sizing for services, estimate the required capacity for individual systems, networks, and storage. Sizing described here only accounts for DS servers. Monitoring and audit tools, backup storage, and client applications require additional resources.

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

For details about how each component uses system resources, see [DS Software](#).

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.

Memory

Directory server memory requirements depend primarily on how you cache directory data. If your directory data set can fit entirely into system memory, provision enough RAM to cache everything. The RAM available for the server should be 1.5 to 2 times the total size of the database files on disk. By default, database files are stored under the `/path/to/openssl/db` directory.

By default, DS directory servers cache database internal nodes in the JVM heap. The file system cache holds the database leaf nodes. For details, see [Cache Internal Nodes](#).

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

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. Be aware 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.

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 [Plan to 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.

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

Deployment Checklists

Use these checklists when deploying your directory service:

Initiate the Project

Task	Done?
Understand the business requirements for your DS deployment	<input type="checkbox"/>
Identify key client applications	<input type="checkbox"/>
Identify project stakeholders	<input type="checkbox"/>
Define SLOs based on business requirements	<input type="checkbox"/>
Define project scope	<input type="checkbox"/>
Define project roles and responsibilities	<input type="checkbox"/>
Schedule DS training for deployment team members	<input type="checkbox"/>

Prepare Supportability

Task	Done?
Find out how to get help and support from ForgeRock and partners	<input type="checkbox"/>
Find out how to get training from ForgeRock and partners	<input type="checkbox"/>

Task	Done?
Find out how to keep up to date with new development and new releases	<input type="checkbox"/>
Find out how to report problems	<input type="checkbox"/>

Design the Service

Task	Done?
Understand the roles of directory components	<input type="checkbox"/>
Define architecture, mapping requirements to component features	<input type="checkbox"/>
Define the directory data model	<input type="checkbox"/>
Define the directory access model	<input type="checkbox"/>
Define the replication model	<input type="checkbox"/>
Define how to backup, restore, and recover data	<input type="checkbox"/>
Define how you will monitor and audit the service	<input type="checkbox"/>
Determine how to harden and secure the service	<input type="checkbox"/>

Develop the Service

Task	Done?
Engage development of custom server plugins as necessary	<input type="checkbox"/>
Apply configuration management	<input type="checkbox"/>
Create a test plan	<input type="checkbox"/>
Engage automation, continuous integration	<input type="checkbox"/>


Task	Done?
Create a documentation plan	<input type="checkbox"/>
Create a maintenance and support plan	<input type="checkbox"/>
Pilot the implementation	<input type="checkbox"/>
Size systems to provision for production	<input type="checkbox"/>
Execute test plans	<input type="checkbox"/>
Execute documentation plans	<input type="checkbox"/>
Create a rollout plan in alignment with all stakeholders	<input type="checkbox"/>
Prepare patch and upgrade plans	<input type="checkbox"/>

Implement the Service

Task	Done?
Ensure appropriate support for production services	<input type="checkbox"/>
Execute the rollout plan	<input type="checkbox"/>
Engage ongoing monitoring and auditing services	<input type="checkbox"/>
Engage ongoing maintenance and support	<input type="checkbox"/>

Maintain the Service

Task	Done?
Execute patch and upgrade plans as necessary	<input type="checkbox"/>
Plan how to adapt the deployment to new and changing requirements	<input type="checkbox"/>

Was this helpful?  

Copyright © 2010-2024 ForgeRock, all rights reserved.